POLITECNICO DI TORINO

Department of Control and Computer Engineering

Master course in Mechatronic Engineering

Master's Thesis

Virtual commissioning of a pneumatic servosystem with a PLC in MiL, SiL, and HiL

Supervisor:
Prof. Luigi Mazza
Co-supervisor:
Eng. Marco Pontin

Candidate:
Francesco Paolo Miseo
229043

A.Y. 2018-2019

Abstract

This thesis work concerns the virtual commissioning for the control with PLC of an inverted pendulum pneumatically actuated. In detail, a model of our system within the *Simcenter Amesim* software was created, which permitted to perform a validation of the code created for the PLC, through the subsequent phases of Model-in-the-Loop, Software-in-the-Loop, and Hardware-in-the-Loop. After the review of the scientific literature, the functioning of the inverted pendulum present in our laboratory was described, analyzing studies already done previously on the system. All the steps that led to the construction of the model within the Amesim software were examined, followed by the programming of the PLC first virtual, then real, and by the solutions adopted to put Amesim in communication with other software and hardware components that have allowed the realization of SIL and HIL. After the validation activity, the test bench was used to test the effective functioning of the created control and to improve the stability of the system by using a Siemens PLC that is more performing than the one already in use on the test bench. Finally, all the simulations carried out both on the virtual system and on the test bench using the models previously created was analyzed and compared.

List of Contents

List of Figures

List of Tables

Introduction

This research objective is to study how to manage the virtual commissioning for the control with PLC of an inverted pendulum pneumatically actuated.

Therefore, we will introduce some theoretical refences on the simple pendulum, and then on the inverse pendulum.

A simple pendulum consists of an inextensible wire to which a material point of mass is fixed below, which can oscillate around a fixed point called pole. In this system, the weight force component along the wire counterbalances the tension of the wire itself, while the weight force component perpendicular to the wire acts as a pull force and produces the oscillatory motion of the pendulum. The inverse pendulum represents a simple inverted pendulum, rigid and without a pole: the lower part can therefore move to balance the oscillations of the upper one in order to ensure balance.

The reason why we have deepened the issue of the reverse pendulum is that, although it is one of the most discussed subjects in the field of control theory, no definitive solution has been found to control it. In this regard, we focused on the study of the model we created, which simulates the physical system, and how close this can be to reality.

In practice, the reverse pendulum model can be found in many applications: from the Segway, an individual means of transport that uses a platform with two wheels that can be swiveled by means of a handlebar, to the maintenance of robots in an upright position and the stabilization of the rockets. Everything we have described needs to remain stable in order to function: it is precisely in this

area that our work is placed, namely the study of the control algorithm that allows to stabilize the pendulum. In this respect, it should be remembered that our work is carried out on the study of a pendulum implemented pneumatically and controlled by PLC (*programmable logic controller*).

Our work is therefore divided into several chapters, which analyze in detail what we have just described.

In the first chapter we briefly explain the state of the art and then the studies from which we have taken as a starting point to continue the research.

Then, in the second chapter, we will describe the test bench in the laboratory and we will analyze the various subsystems (mechanical, pneumatic, electrical) that compose it. Moreover, the linearized problem used to implement the control algorithm in the virtual model is studied again.

In the third chapter, we will describe the first phase of validation (virtual commissioning) carried out through Model-in-the-Loop, in which it is described how the *Simcenter Amesim* software has been used for the construction of our system. Moreover, it is also illustrated how the simulation between *Simcenter Amesim* and *Simulink* has been made.

In the next chapter, we will illustrate the next phase of Software-in-the-Loop that will allow us a first validation of the source code created for the virtual PLC using the *TIA Portal* software, code that will be described in the chapter. We will continue by illustrating the way in which it was possible to put *Simcenter Amesim* in communication with the virtual PLC.

Finally, in the last chapter, we will present the Hardware-in-the-Loop phase, in which the real PLC is used to control the *Simcenter Amesim* model. It will be then deepened how the connection between *Simcenter Amesim* and the real PLC has been made through the hardware platform *SIMIT UNIT*.

1 State of the art

Among the various problems analyzed by the scientific literature, we certainly find the problem of the inverted pendulum, one of the most treated ever. Generally, we deal with the issue of the inverted pendulum in an actuation system that requires its operation an electric engine, therefore, we are often faced with an electric actuation system, precisely.

In the case that we are going to analyze during our thesis work, the actuation system is pneumatic, which makes it a very peculiar system since it uses a fluid with high compressibility such as air.

Many of the choices made during the entire experimentation process are the result of analyses carried out on the state of the art, on which we have based our theoretical and practical studies.

In [1] the construction of the test bench used in the laboratory is explained. The test bench played a fundamental role in our experimentation and reference will be made to the study of the linearized problem carried out within it in order to optimize our controllers.

In Petric et al. [2] it is considered an experimental device made of a cylinder without stem, a 3/2 proportional valve and a pendulum of 400 mm length. The supply pressure used is 6 bar, the mass of the slide to which the pendulum is bound is equal to 1:5 kg, while the latter has a mass of 0:06 kg. This is checked using the LQR method

In Krupke and Wang [3] the control of a pneumatically operated inverted pendulum system is analyzed from a purely theoretical point of view. In fact, the model involves the use of a cylinder without stem and a 3/2 proportional valve with closed centers. The control architecture requires the use of an internal

control loop of the force applied by the actuator, in which a compensator type PI is used, and an external ring for stabilizing the position of the cart and the pendulum, in which a controller type full-state feedback is implemented. In addition, a friction estimator is used to compensate for the variable and non-linear effects of friction.

In Zilc et al. [4] the actuation system consists of a 3/2 closed center proportional valve with a 100 Hz bandwidth and a pneumatic cylinder without stem stroke of 0:5 m and 15 mm bore. The angular transducer is of the rotary potentiometric type, while the linear transducer is of the potentiometric type. System control is first achieved with a linear controller of the state feedback type, to which a non-linear controller is then added to compensate for the effects of friction. More complex control solutions based on LQ and LQG optimization procedures are then adopted.

In the last three cases, there are differences with respect to the system under study. As detailed below, the cylinder used has a stem, resulting in a difference in the surface area of the piston influence. The control strategy is also simpler: in fact, it was decided to use two control rings, one external (to control the position of the cart) and one internal (to control the inclination of the pendulum), each of which uses a compensator type PID. The PID structure is also simple to implement in a PLC.

2 Test bench

As already mentioned above, the system we wanted to model is an inverted pendulum which is present in our laboratory and used for educational purposes.

Below we will list the various characteristics, starting from the mechanical components, through the pneumatic ones and ending with the electrical ones.

2.1 Mechanical components

Among the mechanical components we will discuss in this paragraph, we find first of all the rod: it is a telescopic rod that has a length ranging from 400 *mm* to 700 *mm*, at the tip of which we can insert discs of different weights. In total, the rod does not weigh more than 200*g*, so that the hypothesis of concentrated mass can be guaranteed. The rod is connected to the cart by a hinge, and its total travel is intentionally limited to about 90°. The rod can be easily separated from the cart, which makes it easy to perform tests on the cart if we want to perform a position check.

On the other hand, the cart is connected to a recirculating ball bearing guide anchored to the test bench.

2.2 Pneumatic components

The actuator connected to the cart is a double acting cylinder with a stroke of 500 *mm* and a bore of 16 *mm*, while the rod has a diameter of 6 *mm*.

The solenoid valves are type 2 way 2 position normally closed and are controlled by special drivers that guarantee proportional behavior. The drivers can be controlled either in voltage $0 \div 10$ *V* or in current $0 \div 20$ *mA*. Both the drivers and the valves are powered by 24 *V*.

2.3 Electrical components

The electrical components include two sensors. The linear displacement sensor is of the *LVDT* type and allows to calculate the position of the cart and the relative displacement of the piston. It is connected to the cart by a spherical joint and has the same stroke as the cylinder, equal to 500 *mm*. The voltage output varies in the range 0 ÷ 10 *V*. The other is the angular one and is a Hall effect sensor. It is connected to the shaft of the rod and allows to measure the angular displacement of the latter. It is supplied at 5 *V* and has a voltage output of 0.5 ÷ 4.5 *Vc*, while the measuring range used is 0 ÷ 90°.The last component is a Siemens PLC, which is part of the *S7-1200* family and manages the control of the entire system. In addition, two modules have been integrated, one to manage the analog inputs from the sensors and another for the analog outputs that control the drivers of the valves.

2.4 Test bench operation

In order to control the inverted pendulum, the PLC receives the signals from the two sensors and after applying the control algorithm, it controls the valves through the drivers. As we can see, the valves are two in total and are controlled in pairs, thus using only two analog outputs. When it is necessary to release the rod and move the cart to the right, valves *V2* and *V4* are powered. The first allows the air to enter the rear chamber, while the second allows the air to leave the front chamber. On the contrary, when the piston has to be retracted, it is controlled by the *V1-V3* couple. In this way it is not necessary to check the valves individually.

2.5 Simcenter Amesim

Before testing the code, we created on the real system, we had to consider the following auditing techniques in succession: Model-in-the-Loop (MiL), Software-in-the-Loop (SiL) and Hardware-in-the-Loop (HiL). The study of the system, according to these phases, was carried out using *Simcenter Amesim*, which is a simulation software that permits to model and simulate multi domain problems, as the one we are dealing with.

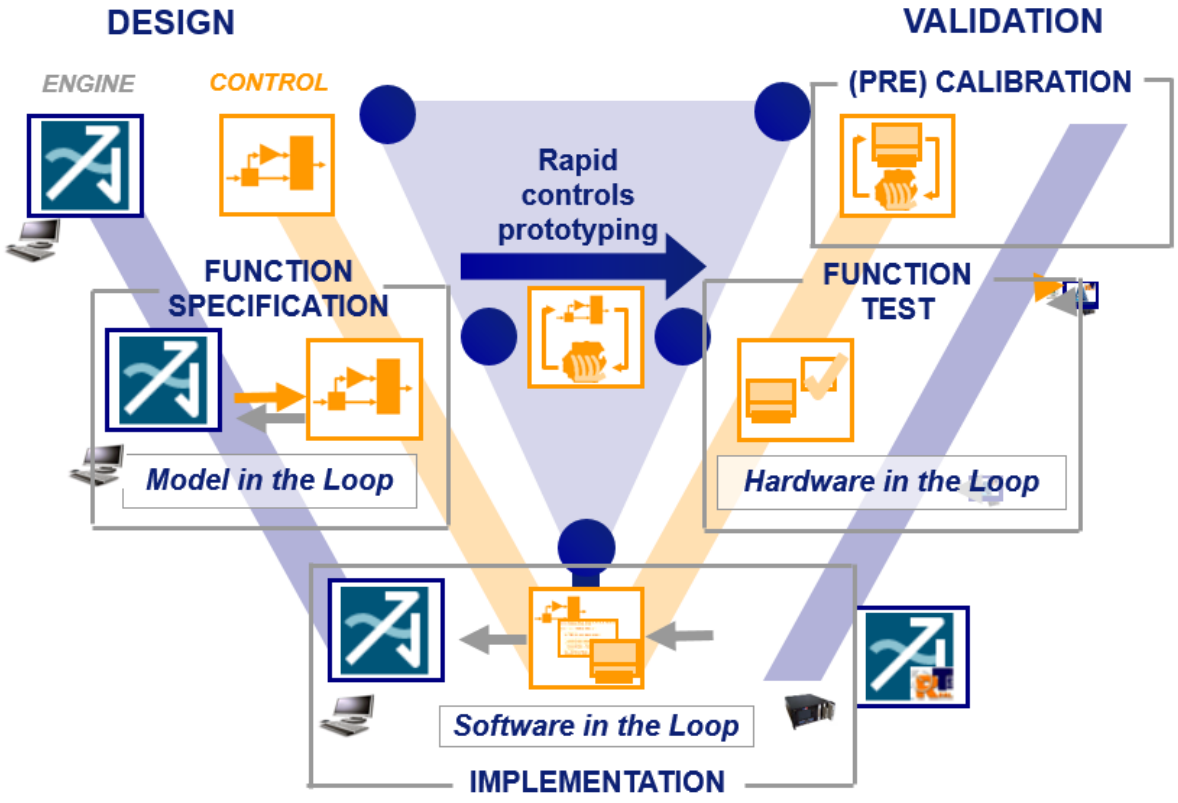


Figure 1: Validation diagram

We tried to create a model as close as possible to reality, then we chose components in the standard library and, for each of these, we have chosen a sub-model closer to the hypotheses made with the data at our disposal.

In *Amesim* there are four main tabs, which are *Sketch*, *Submodel*, *Parameter* and *Simulation*. We started our analysis using *Sketch* and created our model by inserting components from library and connecting them; we continued using the *Submodel* tab: consequently, if all components are properly connected, we have the possibility to pick from a list of available sub-models the most appropriate one. After choosing the sub-model, we set parameters for each element in the relative tab, and, finally, we started the simulation by clicking on the *Simulation* tab to let the software compile the model.

Every component in *Amesim* have a different number of ports that allow to connect each of them to other compatible components of the same domain, and, in certain specific cases, to components of other domains. Each port can host different kinds of variables (force, speed, etc.) both ingoing and outgoing, so if a variable exits from a port, the same variable must entry in the port connected and vice versa. We can see an example in the figure below where the port n. 1 of element *a* is linked to the port n. 3 of element *b*.

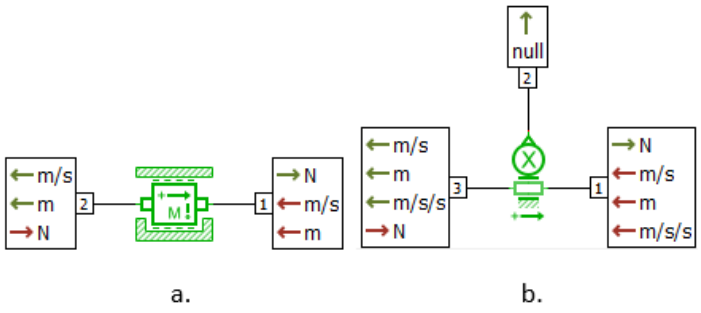


Figure 2: Input/output variables

3 Model-in-the-Loop: MiL

MiL testing is a phase during which system simulation is performed by using a model of the system: it allows a first verification of the requirements and the algorithms of the solution adopted. Even if we can obtain an initial review, during this phase, the forecast of the required hardware processing resources is difficult.

In the following figure, we can see a sampling scheme of our full system in *Amesim*.

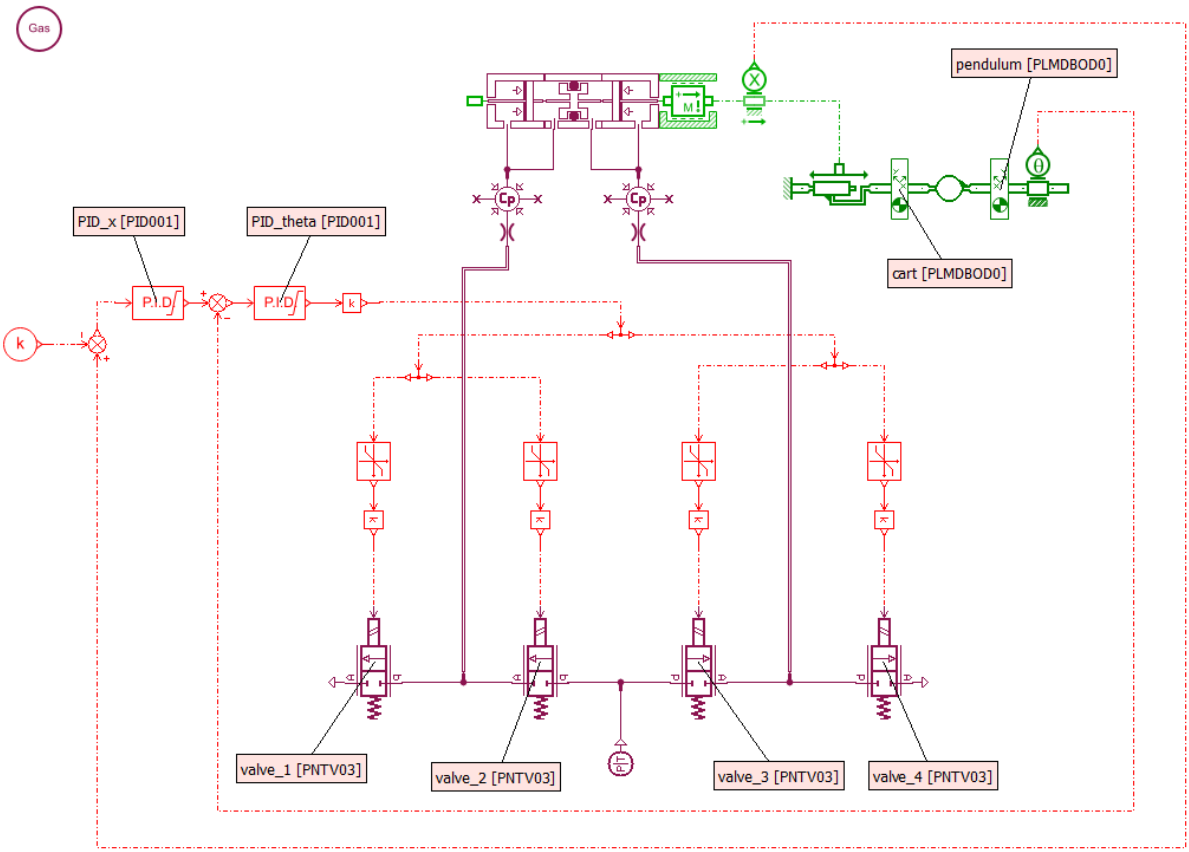


Figure 3: Amesim complete model

3.1 Pneumatic components

The components used are obtained by the standard pneumatic library and they are purple colored. We decided to neglect thermal exchange effects between the components and the environment, and the ones on the working fluid.

3.1.1 Gas model, sources and exhausts

The component in figure below allows us to define gas characteristics of a working fluid. In this case we are using the air assumed as perfect gas; the parameters used are in the following table:

Figure 4 Gas property

Air as a perfect gas	
γ	1.4
R	287.2 J/kgK
μ	$1.82*10^{-5}\ Ns/m^2$

Table 1: Air parameters

The source and the exhaust, respectively *a* and *b* in *Figure 5* are assumed as ideal: they are both at temperature of 293.15 *K*, and, for the source the pressure is 6 *absolute bar*, while for exhausts is 1 *absolute bar* (atmospheric pressure).

a. b.

Figure 5 Source and exaust

3.1.2 Valves

The valves used are, like in the test bench, 2 way 2 position proportional pneumatic servo-valves. The are 4 valves like the component presented in *Figure 6*, whereas their parameters are summarized in the table below:

Valves parameters	
C	14.3 *l/min*bar*
b	0.3
ω_n	50 *Hz*
ζ	0.7

Table 2: Valves parameters in Amesim

The valve natural frequency and the damping ratio were estimated on the basis of similar components, because was not possible to calculate them with experimental tests.

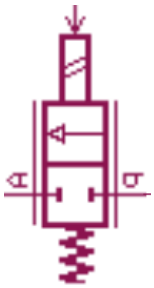


Figure 6: Valve

The valves work in pairs, so a couple lets the piston extend, while the other on allows the piston to retract. We used this working scheme in order to reproduce what it happens in the test bench.

3.1.3 Pneumatic pipes

For the pneumatic pipes between sources, the valves and the cylinder, we chose a sub-model that takes into account the resistance and the capacity: we set only the internal diameter and the pipes length, dimensions are included in *Figure 7*:

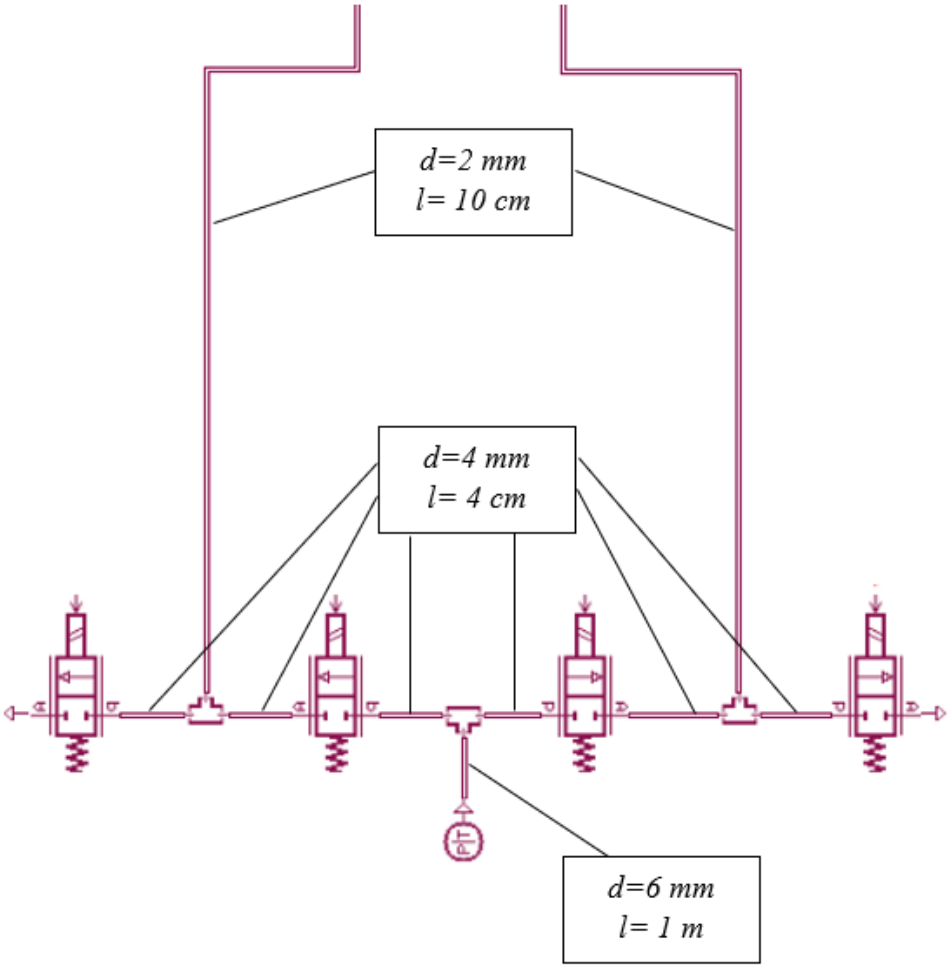


Figure 7: Pneumatic pipes dimension

To connect pneumatic pipes, we used the *T-junction* sub-model only specifying one input and two outputs diameters because of the lack of data.

Figure 8: T-junction

3.1.4 Pneumatic chambers and orifices

The element *a* of the figure below is used to model a pneumatic chamber, therefore, in conformity with our piston, we set dead volumes of both front and back chamber to 1 cm^3, while pressure was 3 *absolute bar*; in addition, these ones were coupled with two orifices as the type represented in *Figure 9-b*, which work like resistances and where we set two parameters: area equals to 1.8 mm^2 and discharge coefficient to 0.4.

Figure 9: Pneumatic chamber and orifice

3.1.5 Pneumatic piston

The pneumatic piston was modelled trough the *Pneumatic Component Design library* that assure a major complexity respect to basic pistons of pneumatic library. As shown in *Figure 10*, the piston consists of three elements: *a* and *c* have the same characteristics and allow to model both rear (*a)* and front (*c*) chamber of the fixed body actuator; what changes is variables associated with ports, which are interchanged.

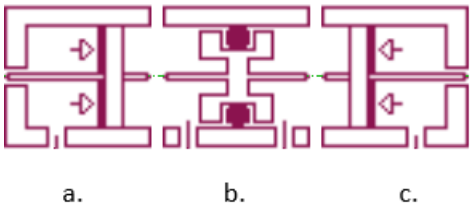


Figure 10: Piston design

Chambers parameters are summarized in the table below:

	Rear chamber	**Front chamber**
D	16 *mm*	16 *mm*
d	0 *mm*	8 *mm*
l	0 *mm*	500 *mm*

Table 3: Piston parameters

The last element *b* of the *Figure 10* is necessary to set piston sealing friction parameters, as follows:

dynamic friction pressure gradient = 8 *N/bar*

dynamic to stiction friction coefficient = 1.4

stick displacement threshold = 0.1 *mm*

equivalent viscous friction during stiction = 1000 *N/(m/s)*

As in the case of the components mentioned above, also with the sealing friction parameters was not possible to execute tests for experimental validation. For this reason, we created a script in *Matlab* that helped us to estimate those parameters. We will describe the process in paragraph 2.6.

3.2 Mechanical components

The mechanical components were picked up from *1D* and *2D Mechanical library* and they are all green colored.

3.2.1 Friction and end stops

Figure 11: End stop

The element in *Figure 11* connected to one of the piston ends, let us introduce stiction, viscous friction and elastic end stops to our system, and, set all the parameters listed in the table. Lower and higher displacements permit us to define stroke of pneumatic jack: even then, we used the *Matlab* script mentioned before to approximate the friction parameters.

End stops parameters	
Lower displacement	0 *m*
Higher displacement	0,5 *m*

End stops parameters	
k_t	1000 N/mm
η	5 $N/(m/s)$

Table 4: End stops parameters

3.2.2 Rigid bodies

By using the two elements represented in *Figure 12*, which are part of planar mechanical library, we modelled the cart and the pendulum. With this component, we could set items location and inertia parameters, so concerning the cart, the mass set up is the sum of the cart mass itself plus piston mass: in fact, it was not possible to consider the cart mass among the elements used for modelling the piston. On the other hand, concerning the pendulum, we considered the barycenter at upper end of rod to simulate a bar without inertia.

Figure 12: Rigid body

In the table is possible to see all data used for modelling the two components:

	Cart	**Pendulum**
G_x	0.25 m	0.25 m
G_y	0 m	0.5 m
x_1	0.25 m	0.25 m
x_2	0.26 m	0.25 m
y_1	0 m	0 m
y_2	0 m	0.5 m

Table 5: Cart and pendulum parameters

3.2.3 Joints

The system represented below contains two joints that are driven by prismatic and revolute pair: the first one is depicted in *Figure 13-a* and assures a translation along a line and replaces the recirculating ball bearing guide of the test bench; the latter is a joint between the two rigid bodies and it is showed in *Figure 13-b*.

For both of them it was possible to define the spring stiffness and the damping coefficient, but, because of information lack, it was only possible to set damping coefficient of prismatic pair to 20.83 *Ns/m*.

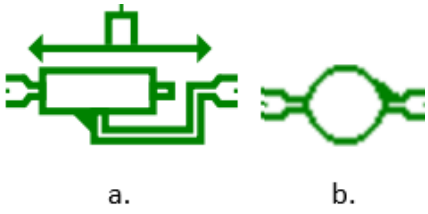


Figure 13: Prismatic and revolute joints

3.2.4 Sensors

The last components of this domain are two displacement sensors: one is linear the other is angular, and, they are respectively elements *a* and *b* of *Figure 14*. The first one is necessary to calculate position of cart, while the latter computes the pendulum inclination.

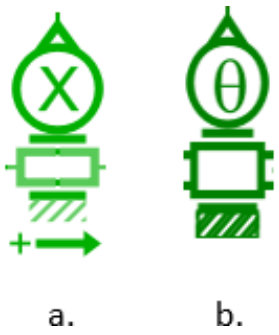


Figure 14: Linear and angular displacement sensors

3.3 Signal components

Components of this library are red colored and the relevant ones for our study are the two represented in *Figure 15-a* and *b*: the PID controller and the saturation elements.

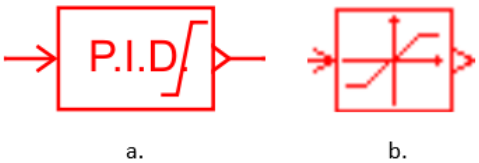


Figure 15: PID and saturation block

The PID controller element was used two times. Beginning from the one in external control loop, input error is the difference between feedback coming from linear displacement sensor and setpoint given by signal reference; its output is compared with angular sensor feedback and it goes into second PID comparator. We can notice how in first PID input error is calculated as feedback minus set and not set minus feedback. Both have outputs limited between -1 and 1, and anti-wind-up method is active: when output reaches saturation, the integrator part receives a zero signal until the PID output goes inside the limits. At the end of the internal PID, there is a gain block that scales signal -1÷1 to -20÷20, that is our rated current valve. Like in PLC, negative signals are directed to valve 1 in order to supply rear chamber and to valve 3 to exhaust front chamber, and, vice versa, if the output is positive, signal arrives to valve 2 and 4 allowing piston to retract.

In this way we can compare the control law used during different phases of validation to the one used during experimental tests.

Two saturation elements were used to filter signals and they are represented in *Figure 15- b*: the first let negative values between -20 and 0 pass and block positive ones, while the second, located to the right, blocks negative signals allowing transition of values from 0 to 20.

Valves work with positive current, so when output signal from PID is negative, a gain block equals to -1, positioned after the saturation element, converts it in positive.

3.4 Co-simulation with *Simulink*

Amesim offers several possibilities to interact with other software: the one that interests us is the possibility of interfacing with *Simulink*

In fact, both the *Amesim-Simulink* and *Simulink-Amesim* interfaces give us the possibility to run different kind of simulations with a combination of the two programs models. Dealing with the two software packages, we have two main options at our disposal: indeed, we can import the *Amesim* model into *Simulink* and the *Simulink* model into *Amesim*.

Furthermore, we can export the *Amesim* solver (with the *Amesim* model) to *Simulink*. The last method in known as co-simulation because software performs the simulation together using the solvers from both packages.

Since certain *Amesim* systems are hard or impossible to be solved using the *Simulink* solver, in cases like the mentioned, we have two choices: we can either use co-simulation (in *Simulink*) or import the *Simulink* model into *Amesim*, exploiting the *Amesim* solver for our intents. However, it happens that the *Simulink* system is very complex and its importation into *Amesim* could not be a good option. Apart from these technical reasons, we must have a clear idea about what the model is used for. If our main goal is either testing or developing the *Amesim* model, the best strategy is to import the *Simulink* model into *Amesim*; vice-versa, in the event that our main purpose is either testing or develop a controller in *Simulink* through a physical model written in *Amesim*, it would be better to work in *Simulink*.

All the working methods available are summarized in table below:

Envisaged interface mode
Co-simulation in *Amesim*: *Amesim* as Master (*Simulink* as Slave)
Model Exchange: Import of *Simulink* model into *Amesim*

Envisaged interface mode
Co-simulation in *Simulink*: *Amesim* as Slave (*Simulink* as Master)
Model Exchange: Export of *Amesim* model into *Simulink*

Table 6: Envisaged interface mode

We have two options for creating an interface with *Simulink*: the standard and the co-simulation interfaces, but there are differences between them. The main one is that the co-simulation interface employs two -or more - solvers contrary to the standard interface that employs only one solver meaning that *Amesim* and *Simulink* use their own solvers for the co-simulation interface but they both use the *Simulink* solver for the standard one. The second difference, instead, is that, using the standard interface, the *Amesim* part is perceived as a time continuous block in *Simulink*, while in the co-simulation it is seen as a time discrete block. The last makes the interface appropriate for discrete controllers implemented in *Simulink* that control an *Amesim* model.

In the figure below, it is possible to see in detail how the interfaces work. In the standard interface, the *Amesim* part of the system gets state variables and input variables from *Simulink*: it is possible to calculate state derivatives and output variables: the *Simulink* solver monitor the entire process of exchanging information.

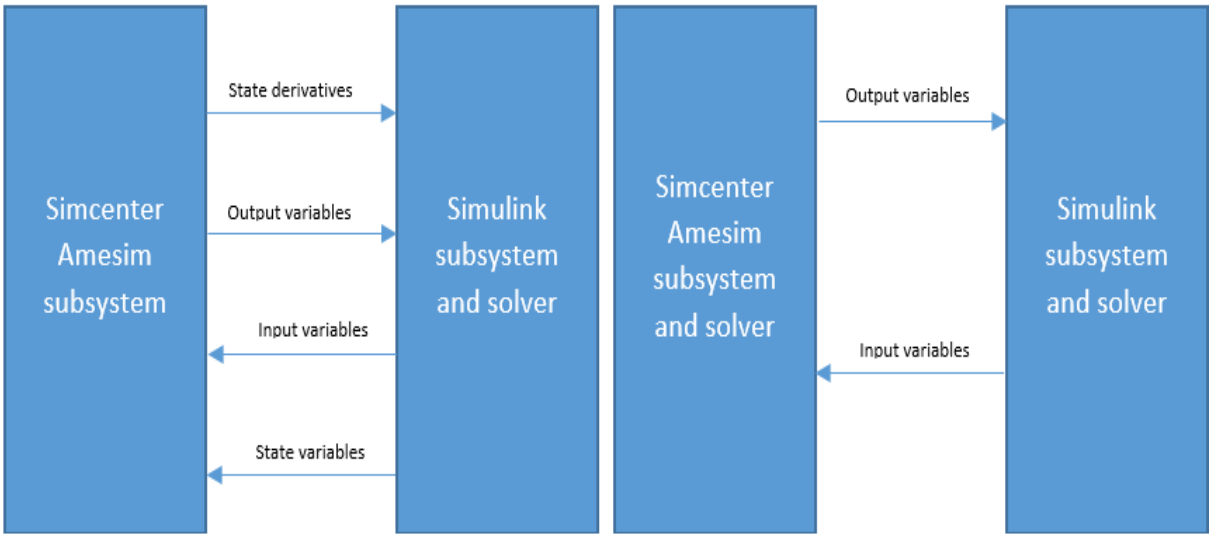


Figure 16: Amesim-Simulink interaction

On the other hand, in the co-simulation, input and output variables are the only ones to be swapped: the rate of exchange is set according to a parameter that we define.

The model is not managed by a single piece of software (*Simulink*), but it is a co-operation among two or more software packages. We must be aware of the fact that there will be an information leak by exchanging input and output variables at a certain sample rate.

We could make a comparison between the process mentioned and the difference of a continuous and a sampled controller.: if we use a smaller sample rate, we get much closer to the continuous result. Apart from this, we must face another problem concerning the leak of information regarding possible cross pairings between the systems – since there is no communication of information on states - and state derivatives.

The *Amesim* to *Simulink* interface let us the building of a model of a subsystem in *Amesim* and its conversion to a *Simulink S-Function*. It is possible to import *S-*

Function into *Simulink* and employ it within a *Simulink* system just like any other *S-Function*.

The interface obtained allows us to continue to use many of the *Amesim* facilities meanwhile the model is running in *Simulink*: we may modify the parameters of the *Amesim* model within *Amesim* in the normal way, examine the results within *Amesim* by creating plots

We can find an example of the process described in the diagram below:

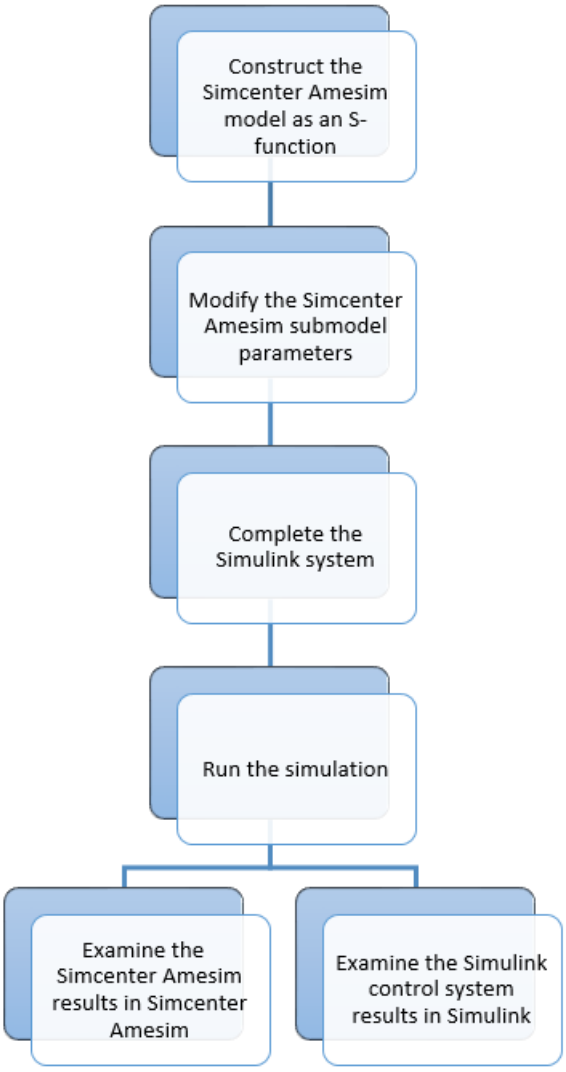


Figure 17: Co-simulation process

Since our goal was to use *Simulink* to manage the control of pendulum, the two-control loop was designed in in the program.

We have reused the system created before in *Amesim* to make co-simulation with *Simulink*, but, this time, PID controllers have been taken off and substitute by an interface block, as we can see from *Figure 18* at the top right. During the creation of an interface block, it can be chosen the type of interface and the number of inputs and outputs. In this case, to make a co-simulation, we chose *SimuCosim* among interfaces, two inputs for linear and angular displacement sensors, and one output for PID signal extracted from *Simulink*. In *Simulink* library, there is an interface block (*Figure 18*) too, named *AME2SLCoSim*; after the importation of the desired model, inputs and outputs defined in *Amesim* appear, but they are reversed.

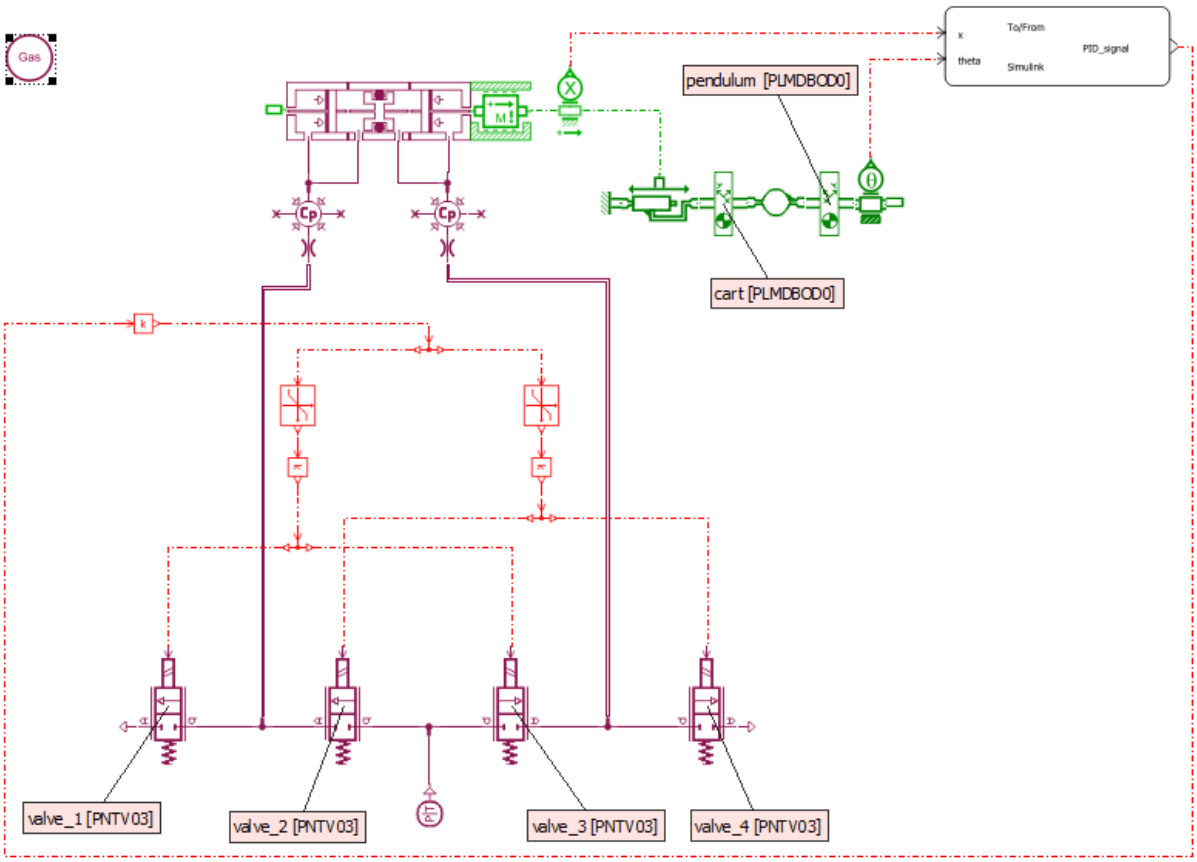


Figure 18: Amesim co-simuation model

By comparing model in *Amesim* and *Simulink*, it is possible to see that what for *Amesim* is an outgoing signal, for *Simulink* is ingoing, and vice versa.

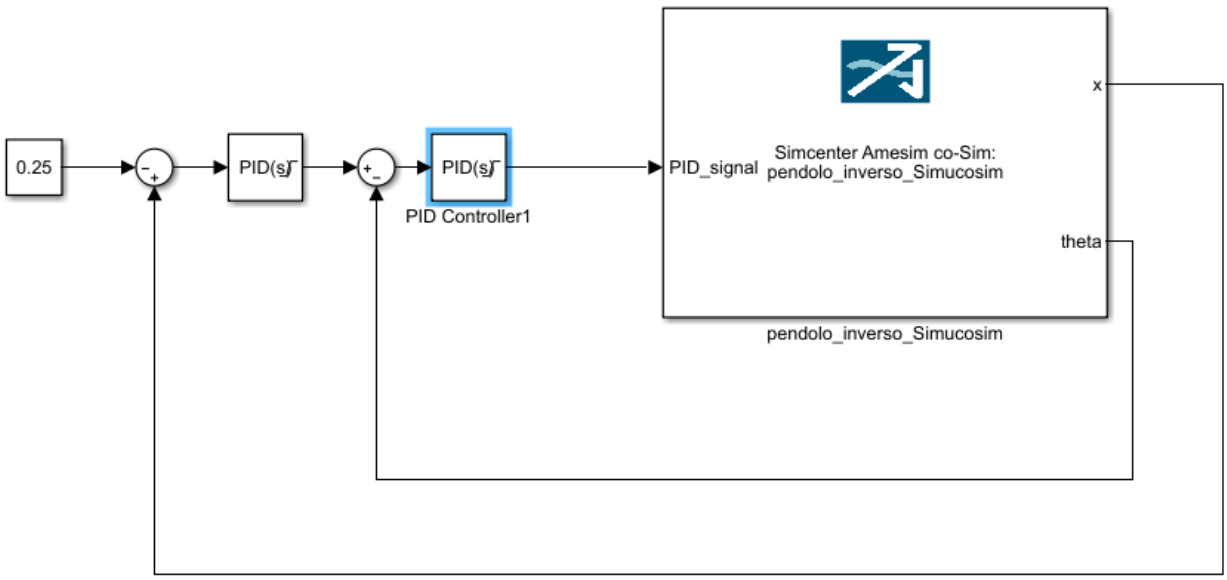


Figure 19: Simulink co-simulation model

3.5 Calculation of friction parameters

Thanks to the type of simulation explained in the previous paragraph, we were able to approximate some parameters of our system, so that the data from the simulation and the data obtained from the experimental tests were as suitable as possible.

In this case, the rod was removed both from our *Amesim* model and from the real system, so as it was possible to get a system with only the cart position control, instead, in *Simulink*, the external control loop of the angle was deleted and only the one with the PID for the linear displacement control remained.

In order to change the parameters of system components, we first had to add the desired parameters to the global parameters table in *Amesim*, then, we created the interface block for the co-simulation in *Simulink*: in the present case, we used the command in *Matlab amecreatecosimsfunmask* to create a block, differently from

what we did before, that also allows to modify global parameters directly from *Simulink*.

All this has been done in order to exploit the script created in *Matlab*, present in the Appendix A. We used the *fminsearch* function present in *Matlab*, which allowed us to find a minimum of unconstrained multivariable function using derivative-free method.

Starting from a starting point x_0, which in this case are our parameters, *fminsearch* calls a function going to modify each time the parameters until it detects the minimum. The function of which we have found the minimum is *opt_par*. This function starts a simulation using the parameters that *fminsearch* modifies at each recall and calculates the Mean Squared Error between the tracking of a square wave signal that we have performed with the test bench, and the same tracking done with the *Amesim* model.

3.6 Simulation results

For each validation phase, simulations were performed to compare the data obtained.

Each test was carried out starting from stationary conditions with the cart halfway up and the pendulum rod in a vertical position. The only exceptions were the simulations of step reference tracking because, given the width of the variation, the stroke of the piston would have been insufficient.

As far as the MiL is concerned, the data were obtained directly from *Amesim*, while for all the other phases, the data were taken through the appropriate tool

in the *Tia Portal* software. Regarding these last measurements, being directly saved on the PLC memory, they have a limited recording time.

In the next paragraph will be shown the results obtained by the controllers optimized from those calculated in [1].

For each validation phase, simulations were performed to compare the data obtained.

3.6.1 Optimized controllers

Using the initial controllers, the system was unstable: this is, probably, due to the fact that some aspects were not considered when studying the linearized problem. After a few attempts, the following controllers have been developed to stabilize the system and ensure a good reactivity.

The controllers mentioned are the following:

$$C_x = k_p + \frac{k_i}{s} + \frac{k_d s}{T_f s + 1} = 0{,}001 + \frac{0{,}0001}{s} + \frac{0{,}055}{0{,}2s + 1}$$

$$C_\theta = k_p + \frac{k_i}{s} + \frac{k_d s}{T_f s + 1} = 5{,}7 + \frac{20}{s} + \frac{0{,}103}{0{,}002s + 1}$$

The optimization of the controllers has been carried out trying to stabilize first the inner ring, that is the angle, and then the outer ring.

The stabilization of the angle is concluded when, starting with the pendulum in a perpendicular position, the cart moves uniformly towards one of the end stops.

The various tests were carried out using a constant signal, a step signal and a square wave signal for the setpoint supplied to the outer ring.

The first graph represents the constant reference tracking with a fixed setpoint at 0.25 m and it can be seen that the pendulum is stable even though there is a continuous oscillation of both x and θ.

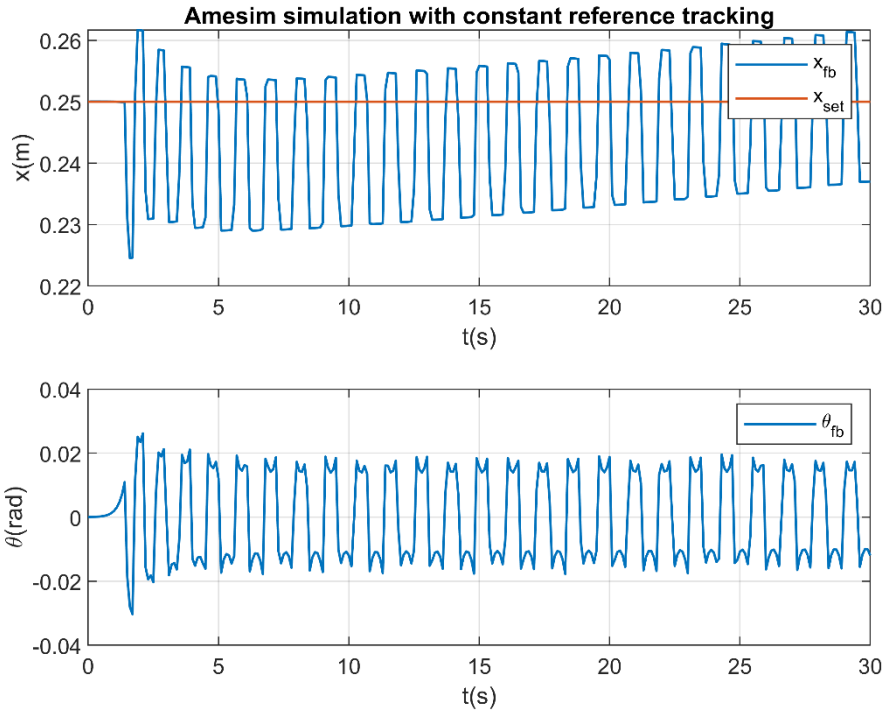


Figure 20: Amesim simulation with constant reference tracking

As far as the step reference tracking test is concerned, it was decided to start the cart from the 0.10 *m* position using 200 *mm* step width. We can notice a sub-elongation typical of systems with no minimum phase rotation, of which the inverted pendulum is part.

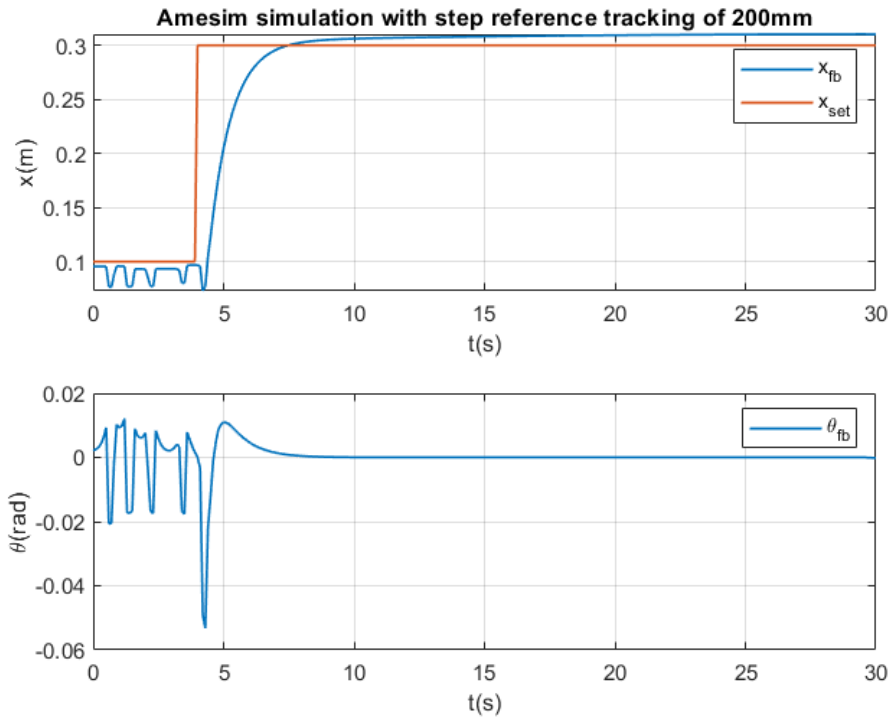


Figure 21: Amesim simulation with step reference tracking of 200 mm

The last test is the tracking of a square wave signal with a period of 8 *s* and a width of 200 *mm*. The cart manages to reach the reference despite some difficulties, while, it is noted that there are no major problems for the stabilization of the angle.

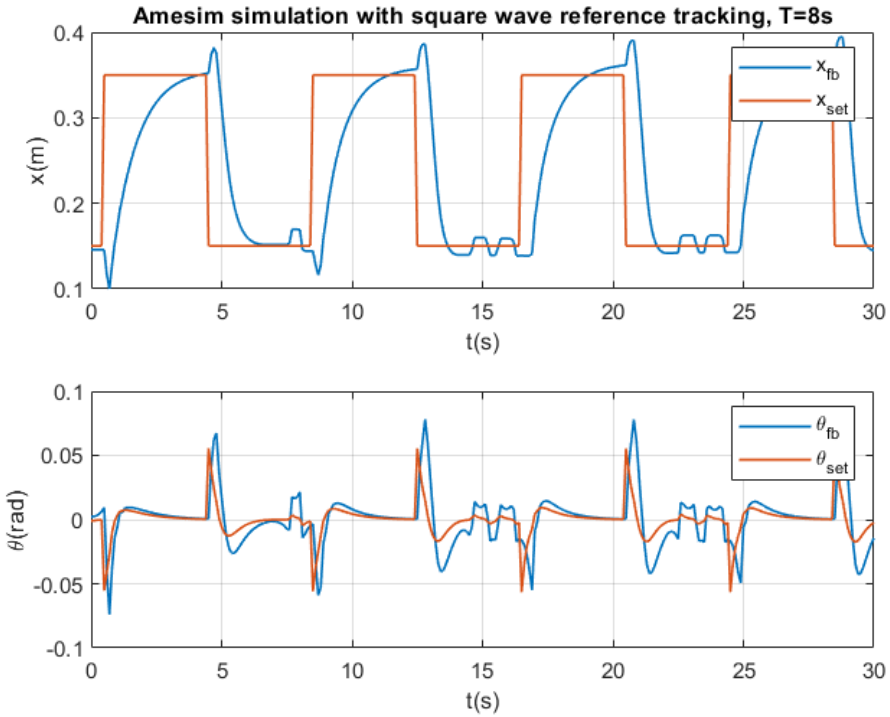


Figure 22: Amesim simulation with square wave reference tracking, T=8s

4 Software-in-the-Loop: SiL

SiL testing is performed by running the software on a normal PC hardware that allows to identify the main errors in the functional domain. However, the compiler and processor of a PC can behave differently compared to the target platform.

We stated below the reasons why SiL is very useful:

- control strategies can be validated virtually, without jeopardizing lives or machines only by using the real PLC code;
- costs can be reduced thanks to the possibility to troubleshoot (it could be late doing the error correction during the design process);
- operators can become familiar with the controller systems, also the ones under construction, thanks to the creation of virtual operator training systems;
- failures can be fixed within few minutes using the "virtual time" capabilities, i.e. acceleration and of time that allow a simulation of a real process in a desired time.

We used at this stage *Automation Connect,* an *Amesim* tool, *PLCSIM Advanced* which is part of the Siemens suite together with the *TIA Portal* used to create the source code of our PLC.

In this chapter we will first explain how the PLC programming was carried out, and then, we will examine how it was possible to exchange data between the different software used.

4.1 Programming PLC

The programming of both the PLCs in the laboratory (*S7-1200* and *S7-1500*) and the virtual ones was made using the proprietary software *TIA Portal V15*, which allows us to interface with the various PLCs and to load the program.

Subsequently in this thesis, we analyzed the structure and described the program we created, which source code is located in the Appendix: we deliberately chose to minimize the complexity of the program by including only the features we needed.

4.1.1 Program structure

Usually, the program of a Siemens PLC realized through *TIA Portal* is characterized by the presence of blocks, four types of blocks to be precise: *Organization block* (purple colored), *Function block* (blue colored), *Function* (green colored), and *Data block* (blue colored).

The *Organization Blocks*, *OB*, define the structure of the user program and are divided into subcategories that allow to decide what the start event will be. This type of block also includes the Main block, which is present in all *TIA Portal* programs and allows to call subprograms one after the other that execute defined subtasks.

Function Blocks, *FB*, are code blocks that store their values permanently in instance data blocks, so that they remain available after the block has been executed.

Functions, *FC*, are code blocks or subroutines without dedicated memory, and are very useful because they can be called several times within the program.

Data Blocks, *DB*, are used to store program data and thus contain variable data that is used by the user program. Global data blocks store data that can be used by all other blocks.

It is necessary to specify that variables can be saved not only within *Data Blocks* but also within tag tables. In this case it is necessary to assign them an address inside the CPU. So, we can divide these variables into three categories: *Inputs, Outputs and Merkers*. As far as the inputs and outputs are concerned, we will have to assign the address associated to the used channel, while for the *Merkers* we have, depending on the CPU that has been used, a series of addresses available. The address is formed in such a way that the first letter indicates the type of operand (I for *Inputs*, Q for *Outputs* and M for*Merkers*), the second letter indicates the type of data and finally there is a number that represents the starting address. Depending on the type of data chosen for a given variable, it will occupy a different number of bytes. In this paragraph we chose to analyses the program we used on the two CPUs in the laboratory, which contains some additional functions, which were not necessary with the virtual model being easier to manage. Within the program there are three *OB*, three *FC* and one *DB*. The language we decided to use is Ladder, very common for PLC programming.

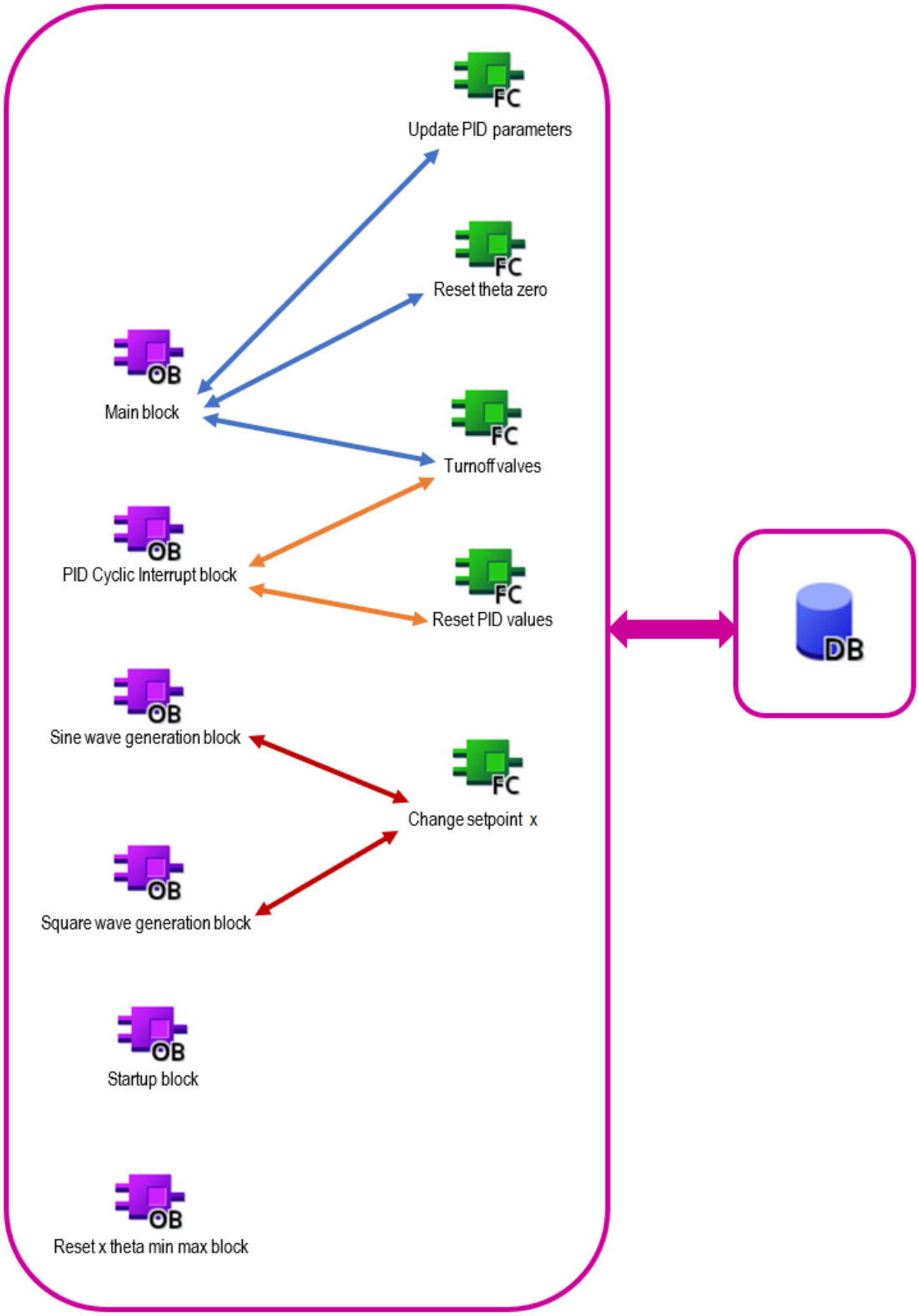


Figure 23: PLC program structure

4.1.2 Main block

The Main block is the first block to be called at each scan cycle and consists of a few networks of Ladder code. The first network is used for the interaction with Amesim, so that when the simulation is started the program can also start on the CPU.

The second network is used to activate theta offset calibration, by calling the appropriate function.

In the third network it is possible to enable the function to modify the parameters of *PID_position* and *PID_angle*.

Afterwards it is possible to activate the cart movement by disabling the two main PIDs and activating one for position control only.

In the fifth and sixth segments it is possible to enable the generation of square and sine waves.

Finally, in the last network we have included the control of various operands that, if activated, disable the operation of the PIDs.

4.1.3 Startup block

The Startup block is of the OB type and is called every time the CPU switches from the *STOP* state to the *RUN* state.

There are only two networks that call two functions: the one that resets the PID values and the other that calibrates the maximum and minimum values of the sensors. This ensures that both functions are called up every time we start the CPU.

4.1.4 PID Cyclic Interrupt block

The *Cyclic Interrupt PID block* is the most important one because it contains the control algorithm. It is part of one of the subcategories of the *Organization Block*, precisely that of the *Cyclic Interrupts*, which allows to start the program at periodic intervals, regardless of the execution of the cyclic program. In our case, we set the interval to $500\mu s$. Inside the block there are the two PID controllers, with the structure used previously, that is double closed loop.

Inside the *TIA Portal*, we can find a PID controller among the technological objects, which is called *PID_Compact*. In order to make the best use of the object, it must be inserted inside a *Cyclic Interrupt block*. This means that the *PID_Compact* will be called cyclically with an interval equal to the one of the OB that contains it.

Once the component has been inserted in our program, we go define the operating parameters. Among these, we can set the limit values for the input and output of the PID, the proportional, integrative and derivative constants and the sampling time. This last one is very important because it represents the interval of time that the PID must wait before updating the output; Ideally it should be set equal to the cycle time of the block that contains it, but since the controlled system needs a certain amount of time to respond to changes in the output value, it is not advisable to update the output at each cycle. For this reason, since the cycle time of the entire block is $500\mu s$, we set the PID sampling time to $1ms$. This means that the PID does the calculation twice before updating the output: this avoids clogging the bus and gives the system time to implement the signal.

PID_Compact is a PID compensator with anti-windup that uses the following control law:

$$y = K_p[(b * w - x) + \frac{1}{Ti} * s(w - x) + \frac{T_d * s}{a * T_d * s + 1}(c * w - x)$$

Below is a description of the symbols:

y	Output value of the PID algorithm
Kp	Proportional gain
s	Laplace Operator
b	Component weighting P
w	Setpoint
x	Instantaneous value
T_I	Integration time
T_D	Derivative time
a	Coefficient for the derivative delay (derivative delay T1 = a × TD)
c	Component weighting D

Going back to the description of the program, in the first two networks we find the two compensators. Again, on the first network, we have inserted a branch with another PID controller that is activated by the *BIT X_CHANGE* and that we have used to perform only the control of the position of the cart. Before x and ϑ become compensators feedback signals, their values are transformed from the range 0 ÷ 27468 to that 0 ÷ 0.5 through a box *NORM_X*, which allows us to normalize the input value that, subsequently, is scaled to the desired range

through a box *SCALE_X*. Through the operations we obtain what will be the feedback of the first PID controller.

It should be noted that when using analog inputs, the PLC processes these values through an analog-to-digital converter, transforming the voltage or current value into a decimal value. In our case, we used the voltage measurement method with a voltage range between 0 and 10V, as the sensors we had, worked in this range. Automatically, the voltage range is converted to decimal, with a resolution of 2^{15} where 0V is equal to 0 and 10V is equal to 27468.

The values x and ϑ were converted in order to obtain a range of $0 \div 0.5m$ for the first and $-0.78 \div 0.78rad$ for the second. Those conversions were done through a box *NORM_X*, which allowed us to normalize the input value that, subsequently, we scaled to the desired range through a box *SCALE_X*. We used the values obtained as feedbacks for the two PID compensators. As for the setpoints, the first PID receives a value in meters that indicates the position of the cart, while the second receives the output value of the first PID.

Concerning the PID for position control, it should be noted that if we want the control to work properly, we must reverse the control logic within the settings of the *PID Compact*. In the first network there is another branch, which is enabled by the open contact *X_CHANGE*, in which we have inserted a PID that allowed us to check only the position of the cart and move it along the guide in the desired position.

The third network is activated if the cart is in one of the end stops with the rod tilted outwards, going to activate the operand that resets the PID controllers, and then, the integral sum, and the function *SHUTOFF_VALVES*. Resetting the PIDs is necessary because if we need to take back the pendulum control, the accumulated error must first be zeroed.

The valves power supply is managed in the fourth and fifth networks: depending on the sign of the PID value, it is directed towards one or the other pair of valves after being properly converted; at the same time, the pair of valves that is not powered is reduced to a minimum value equal to about $5V$ that does not completely close the flow and allowed us to obtain a more linear behavior and a greater reactivity, bypassing the deadband region.

4.1.5 Reset min max values block

The *Program Cycle* OB recalibrates the minimum and maximum values for x and ϑ. The first network is activated after the consents have been checked and the *V1-V3* valves, which allow the piston to retract, are powered.

At the same time, the values of x and ϑ are saved in *X_MAX*, *THETA_MIN*. Timer *T_MAX* after 5 seconds enables output Q and sets *BIT RETRACT*. In this way network 1 is disabled and network 2 is enabled: *V1-V3* are switched off and *V2-V4* are supplied in order to extend the piston, and the values of x and ϑ are saved in *X_MIN* and *THETA_MAX*.

The network remains active for 5 seconds until the *Delay timer* is enabled setting *MANUAL_CHANGE_X* and resetting *RETRACT*. The next network is enabled for 5 seconds that keeps *MANUAL_CHANGE_X* active, allowing the cart to position itself in the center: then, it is reset.

4.1.6 Square Wave Generator block

The block under discussion is also part of the OB and it is of the *Cyclic Interrupt* type. We have set the recall interval to $1ms$. In the first network there is the control of the *Start_SqW operand*: if this is not active, the instruction "skips" is activated,

and it only executes the last network that, when enabled, calls the *Change_setpoint_x* function.

In the second block, there is a timer used as a delay for the start of the square wave signal. The square wave signal with a duty-cycle of 0.5 is generated in the next two networks. In the first network the maximum square wave is imposed at *set_point* with the variable *STEP_UP* for the desired cycle time by using the *TIMER_POS* block.

The Q output of *TIMER_POS* is activated if the latter is powered for a duration equal to that set on *PT*: in this case *Cycle_time_SW*. Once *TIMER_POS* is activated, in the succeeding network, the value of the *set_point* variable is replaced with the minimum value of the square wave, and at the same time *TIMER_NEG* is powered, which activates the output Q after a time equal to *Cycle_time_SW*, disabling for one cycle the upper network, and consequently the lower one. At the next cycle the network 3 is re-enabled and the high value is passed again to set_point, so that, the whole procedure can be repeated until *Start_SqW* is not reset.

4.1.7 Sine Wave Generator block

The block is of the same type as the previous one and has the same cycle time: it is used to generate a sine wave. The first network is responsible for checking whether the generation of the sine wave has been requested through the appropriate operand. Until the operand is set, for each program cycle all instructions will be skipped going directly to the last network. On the second segment there is a timer that restarts at each cycle time set for the sine wave. In the third and fourth segment, the *ET* variable is taken, which is equal to the time elapsed since the timer was started, and the value is transformed into seconds.

Finally, as can be seen from the formula below, it is determined the value to be passed to the *set_point* variable:

$$setpoint = 0{,}25 + Asin(\omega t)$$

The last network resets the setpoint to the initial value when the generation of the sine wave is no longer required.

4.1.8 FC Change setpoint x

It is a very simple function block consisting of a single network that, when called, allows to manually change the setpoint of the PID for position control.

4.1.9 FC Reset PID values

This function is used to set the output values of the PIDs to zero using the *MOVE* element, and to reset them by enabling the *RESET* operand.

4.1.10 FC Reset Theta zero

Sometimes it is necessary to recalibrate theta zero, that is, the input coming from the angular sensor when the rod is in a vertical position: this can be done by calling this function. First the valves are turned off so that the cart does not move, then, in the second network, the value coming from the angular sensor, after being properly scaled, is copied to the variable *THETA_ZERO* that contains theta offset value.

4.1.11 FC Reset x theta min max

It may be necessary to calibrate the minimum and maximum values of the two sensors' signals. To do this, the *V1-V3* valves are first supplied so that the piston retracts to the left end stop, and with the cart in this position and the rod tilted to

the left, the values of *X_MAX* and *THETA_MIN* are changed. After 10*s* from the activation of the first network, timer T_MAX activates the output Q that sets the *RETRACT* operand and disables the first network. The same operand enables the second network in which the piston is extended by turning on the valves *V2-V4*, then the values of *X_MIN* and *THETA_MAX* are rewritten, with the cart positioned on the right end stop and the rod tilted to the right. This network is also disabled after 10s by the Delay timer that resets the *BIT RETRACT* and set the *MANUAL_CHANGE_X*, which allows the cart to move towards the center. From the setting of this last *BIT* we obtain the activation of the last network that only maintains the *BIT* active for 10*s*.

4.1.12 FC Turnoff valves

The function is made of a single network which, when called, does not supply the valves using two *MOVE* blocks to set the outputs to 0.

4.1.13 FC Update PID parameters

The last function changes the parameter values of the two PIDs. When it is called, even when the program is running, it is possible to manually change the values of *Kp*, *Ti* and *Td* for both *PID_position* and *PID_angle*.

4.2 Automation Connect

Automation Connect is a tool that allows the connection of *Amesim* models to different types of real or virtual automation controllers. It is designed to permit a fast and easy interface to map and exchange variables. We will describe two main families of use-cases: *Hardware-in-the-Loop* (HiL) and *Software-in-the-Loop* (SiL). Furthermore, it is possible to combine HiL/SiL cases, that is, cases where

automation hardware is only partly simulated. The following figure shows some HiL and SiL modes supported by *Automation Connect* and employed in our work.

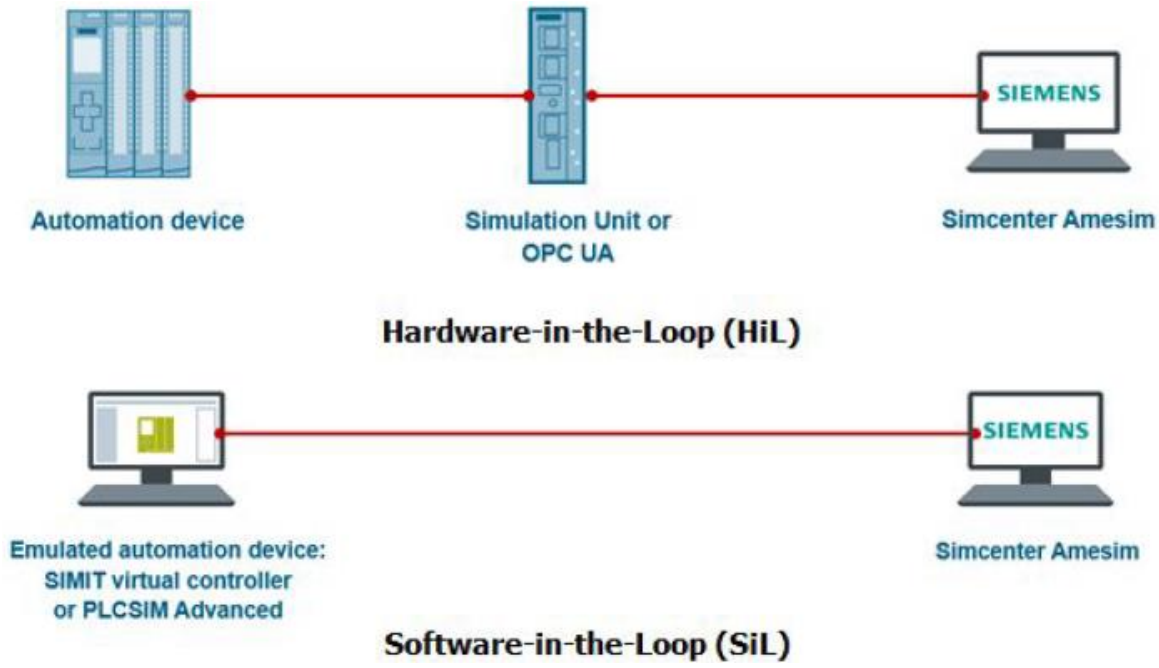


Figure 24: Automation connect supported modes

4.2.1 User interfaces

In this paragraph we will talk about the use case of SiL in *Automation Connect*, anticipating that we will deal with the HiL in the chapter devoted.

Among the various interfaces of *Automation Connect* we used the one with *PLCSIM Advanced* (add-on of *Siemens TIA Portal*): it permits to connect *Amesim* with the emulated recent *PLC S7-1500 series*. We will deepen how *PLCSIM Advanced* functions later in the chapter.

The data exchange between *Amesim* and *Automation Connect* is only possible after inserting the right interface block in the model *Amesim*: we used the same

interface block and chose *Automation Connect* without modifying the system created for co-simulation with *Simulink*.

The tool has several tabs, each dedicated to a different software with which it interfaces. In order to create a connection between *Automation Connect* and our *Amesim* model, from the *Amesim* tab, it is necessary to select the *DLL* file in the base directory, created after compilation. In fact, after the uploading process, the tool lists the defined interface-variables that per definition are of type *Double* (or *LREAL*), meaning a *4-byte floating-point number*.

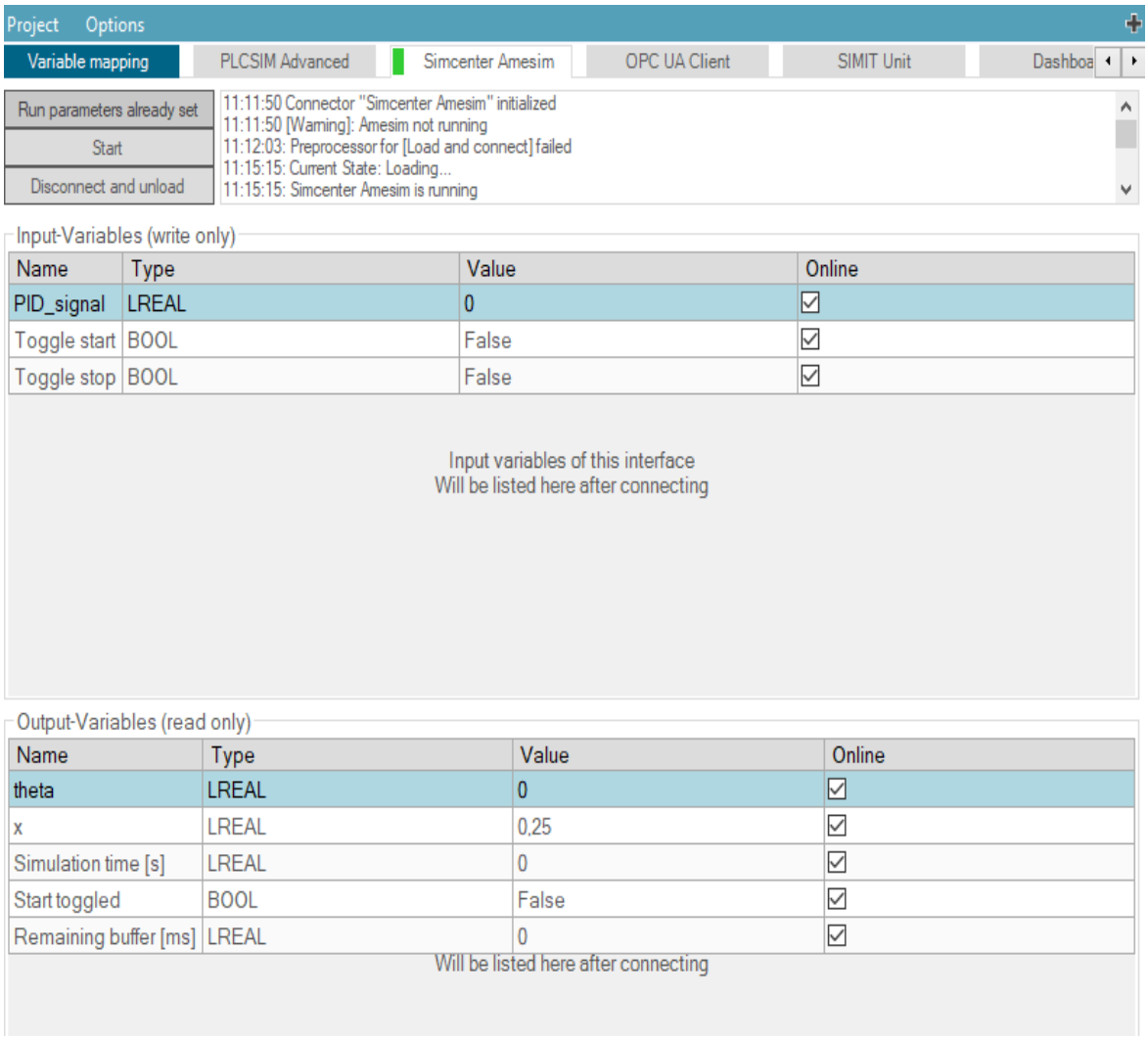


Figure 25: Automation connect Amesim tab

As shown in *Figure 22*, we have at the top the input-variables and at the bottom the output-variables. In addition to the variables in the interface block we defined, the tool always adds one more input-variable, *Toggle start* and two more outputs, *Simulation time* and *Start toggled*. Those can be used by an output from another system to automatically start the *Amesim* simulation. Furthermore, it allows a programmed feedback describing the simulation status and the wall-clock-time synchronization capabilities.

Automation Connect acts as a co-simulation (and time) master and *Simcenter Amesim* as a slave with which it is possible to control the *Amesim* simulation: it is input variables can be inputs to the connected system, whereas output variables can be a feedback given by the connected system: the first can be written, while the latter can be read; it is also possible to write the input values instead of connecting the variables to other systems.

Furthermore, *Amesim* uses the simulation parameters, previously defined inside the tool.

Simulation Interval value defines the timeframe of each simulation macro-step in *Amesim* and it restricts the minimum time for exchanging and updating variables with *Amesim*. During the simulation it is possible to see what is going on in *Amesim* in real time, and plot graphs.

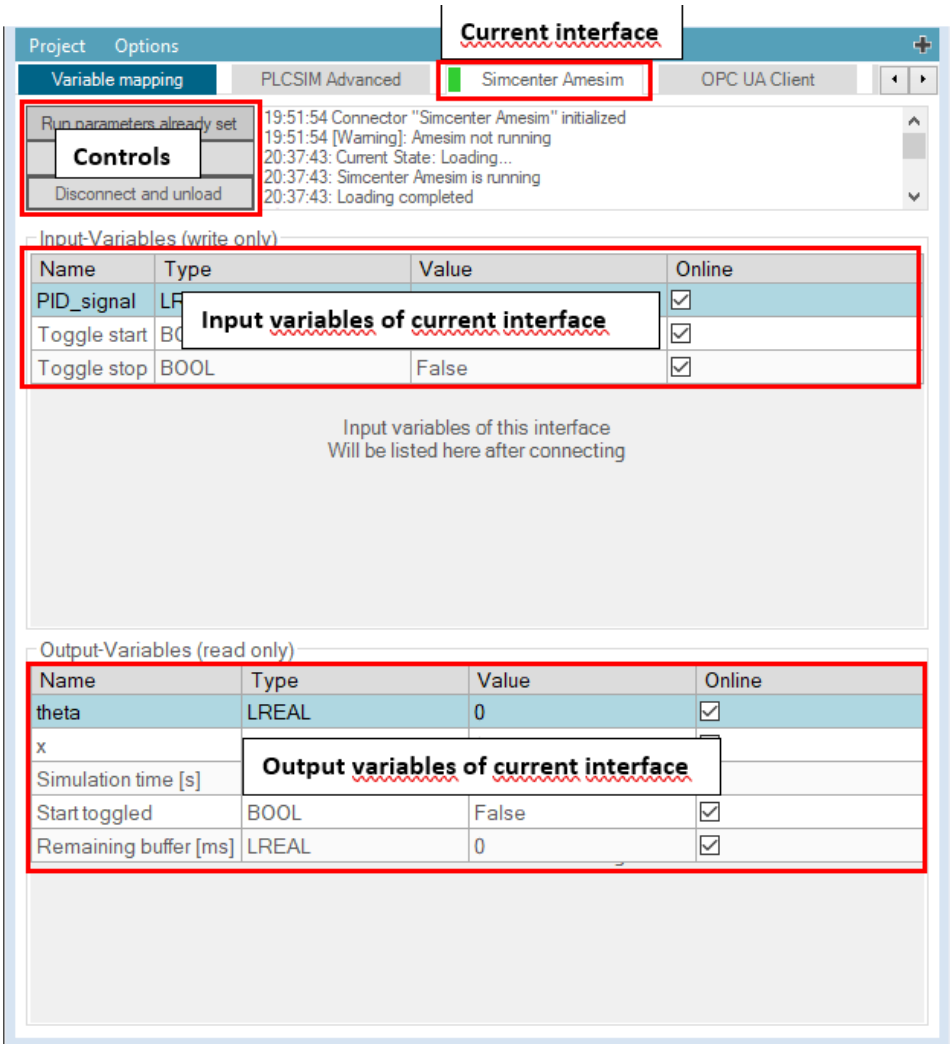


Figure 26: Automation Connect control panel

Since we worked with *Siemens S7-1500 PLCs* and *TIA Portal*, we could emulate our target PLC using *PLCSIM Advanced*: the software can be then connected to *Automation Connect*. In the related tab we can choose from the various existing instances of *PLCSIM Advanced*, and, once we select the one we preferred, a window opens, like the following figure.

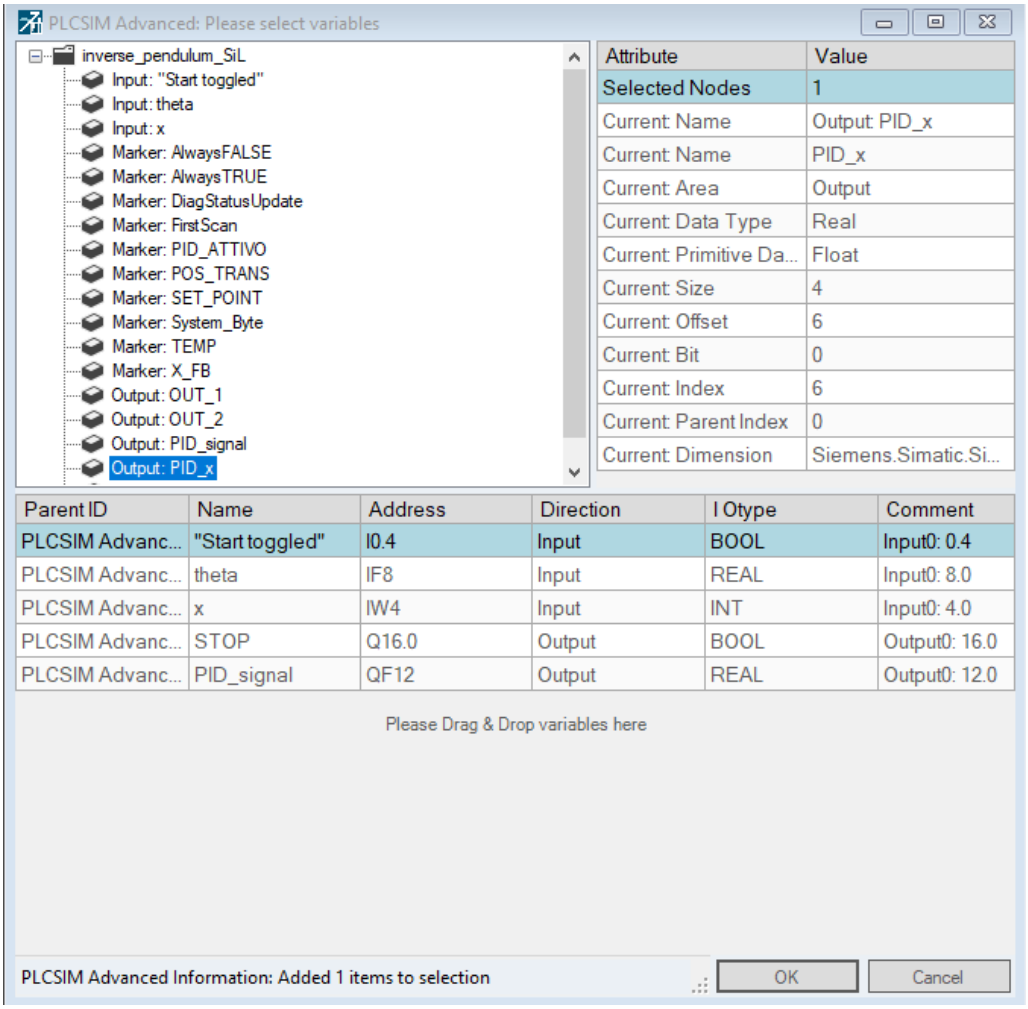


Figure 27: Automation Conncect PLCSIM Advanced tab

This window shows all the variables we've created within the tag table of our PLC program. We can select those of our interest, that is, those that will exchange data with the other variables present in *Amesim*.

For each variable we are shown the direction – either input or output-, address and type: The *Markers* are considered both input and output.

The last step is to connect the *Amesim* variables with those of *PLCSIM Advanced*. To do this we moved to the variable mapping tab, and, connected the input (on the right) of a system with the output (on the left) of the other and vice versa. For

each variable it is indicated which software it belongs to, the name and type. In the Figure are represented the connections that concern our simulation.

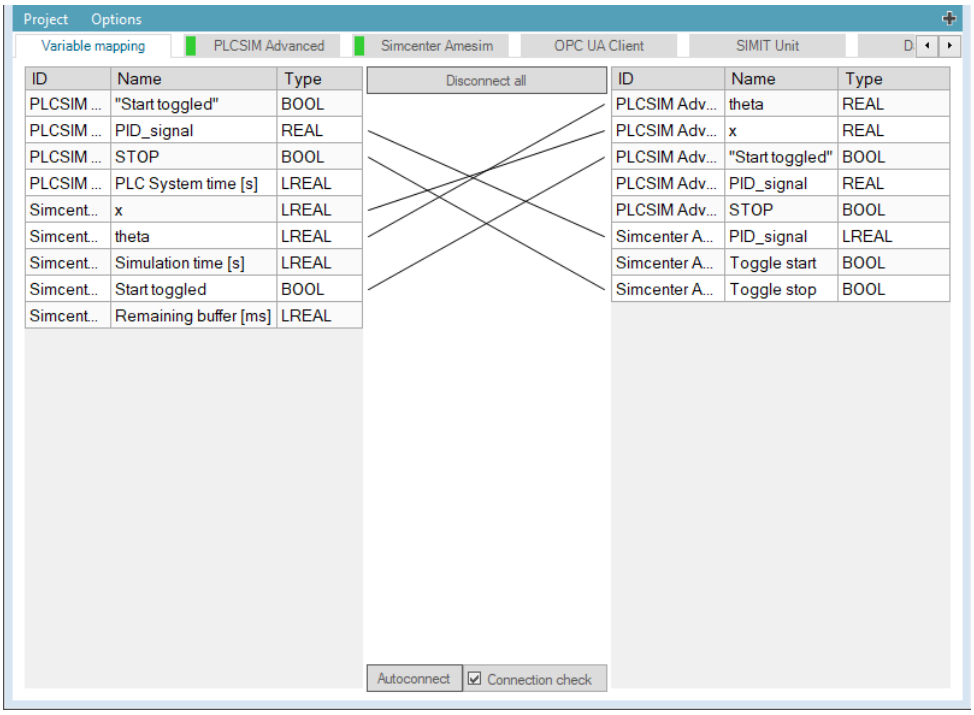


Figure 28: Automation Connect Variable mapping tab

At this point it is possible to start the simulation directly from the *Automation Connect* interface by going to the *Amesim* tab. As far as the virtual CPU is concerned, it will already be in the *RUN* state and ready to control the system. In the source code we have created, every time we start the simulation from *Automation Connect*, thanks to the *BIT Start toggled*, the required networks are activated so that the control can start. The simulation started following these steps has the problem to be asynchronous, but however we have succeeded in stabilizing the system. In any case, the tool offers the functionality to synchronize the various systems and this allowed us to obtain more reliable simulations.

4.3 PLCSIM Advanced

Using *PLCSIM Advanced* we simulated the program for *PLC* on a virtual controller, without needing a real one. We just configured our CPU in *TIA Portal*, we created our program and then we loaded the hardware configuration and program inside the virtual controller. So, we started the CPU and observed the effects on simulated inputs and outputs.

The virtual controller could not fully simulate a real CPU, because even if there were no errors, we could not be sure that the virtual controller would behave exactly like the real one. The scan cycle time and the exact time of actions in *PLCSIM Advanced* were not the same as when these actions run on physical hardware: this is because there were several programs share the processing resources on the PC.

4.3.1 Communication paths

PLCSIM Advanced have two different paths for communicating with *TIA Portal*: Local and Distributed communications. Local communication is carried out via *Softbus* in *PLCSIM Advanced* by default, either making it impossible downloading data accidentally to a hardware CPU or communicating with real hardware.

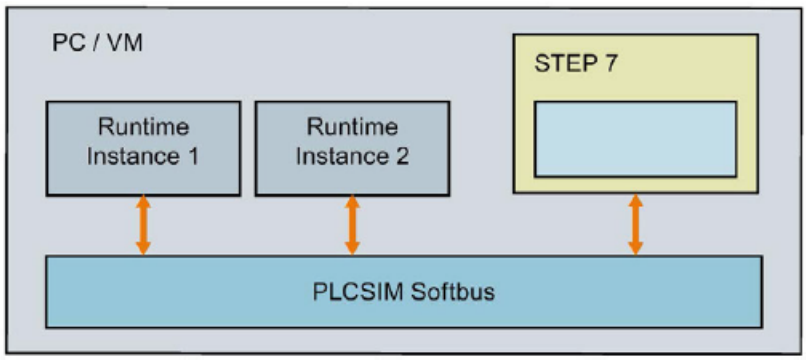


Figure 29: Local communication via Softbus

The second protocol, which may be employed for local communications is *TCP/IP*. Communication is performed via the *PLCSIM Virtual Ethernet Adapter*, which is a virtual network interface behaving as it was a real interface.

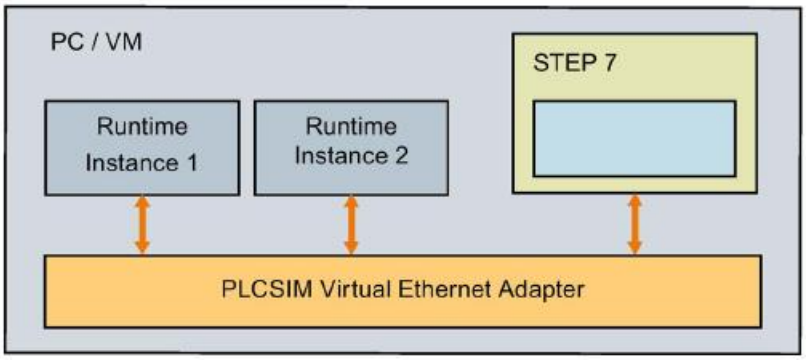


Figure 30: Local communication via TCP/IP

Distributed communication also uses *TCP/IP*: so that *PLCSIM Advanced i*nstances are able to exchange data with other devices through the *Virtual Switch*, and they make also possible communicating with real or simulated CPUs.

In the figure below, we can see the communicative structure of *TIA Portal*, located on one PC, and *PLCSIM Advanced* instances located on another one.

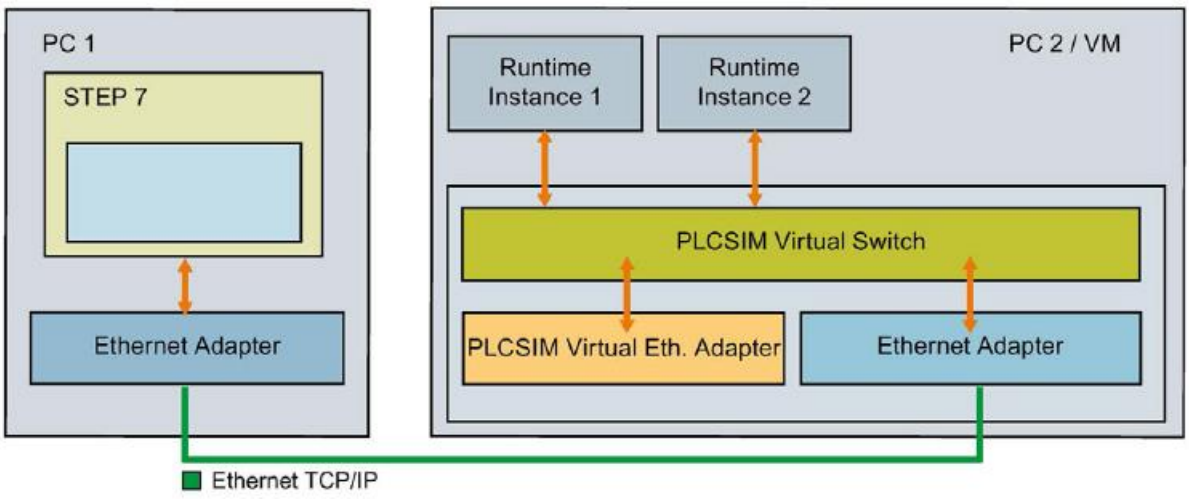


Figure 31: Distributed communication via TCP/IP

4.3.2 Control Panel

In this paragraph we are going to talk about the *PLCSIM Advanced* control panel: in the figure below, we can see an image of how the panel appears.

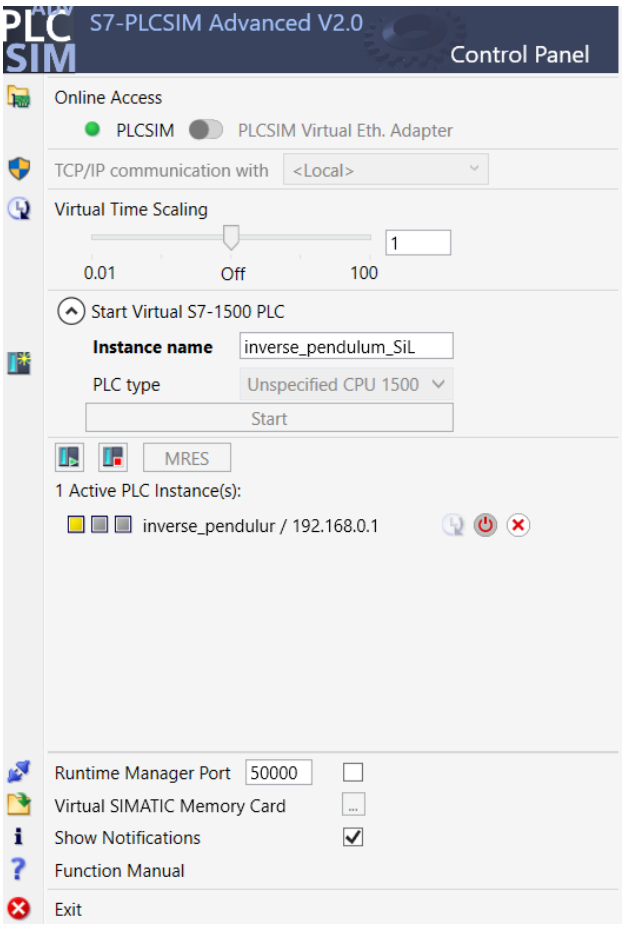


Figure 32: PLCSIM Advanced control panel

Our first step was the choice of the communication type, that is to say *PLCSIM*, a local communication via *Softbus*; it is possible to change the *Virtual Time Scaling*, so the system time runs correspondingly faster or slower.

Afterwards, we defined the instance name so that our virtual controller could start: we will be able to select within the *TIA Portal* the virtual PLC could be created and used as if it was a normal CPU going to load our program.

4.4 Simulation results

Concerning the simulations carried out in SiL, the first step was to start with the controllers used in the *Amesim* model, and then to arrive at the following controllers that have assured a better behavior of the system:

$$C_x = k_p + \frac{k_i}{s} + \frac{k_d s}{T_f s + 1} = 0{,}003 + \frac{6{,}12 * 10^{-5}}{s} + \frac{0{,}132}{0{,}1584 s + 1}$$

$$C_\theta = k_p + \frac{k_i}{s} + \frac{k_d s}{T_f s + 1} = 5{,}1 + \frac{10{,}2}{s} + \frac{0{,}153}{0{,}004 s + 1}$$

As for constant reference tracking, the system is stable, but the behavior is different from that seen in the MiL. As it can be seen, the cart has some difficulty in staying close to the setpoint: this may be due to the step target set within *Automation Connect*. In fact, the tool can have some problems using a target step below 10 *ms*, as in our case. All this goes to affect the data transmission between Amesim and the virtual PLC causing the problems we mentioned above.

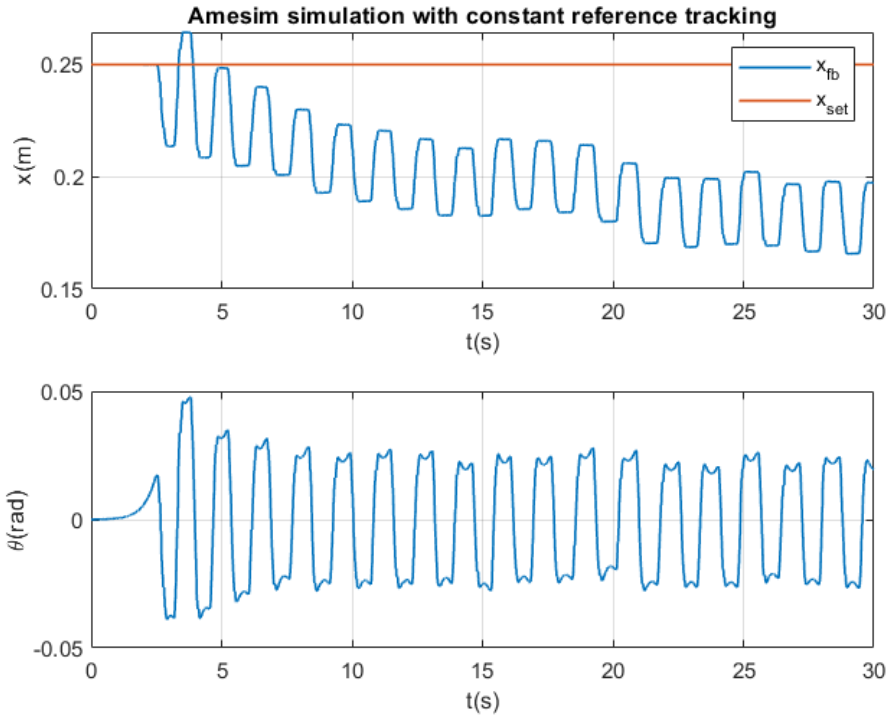


Figure 33: SiL Amesim simulation with constant reference tracking

As for the step and the square wave reference tracking, the behaviors are more similar to those seen before, even if in the first case the system takes a long time to reach the target after the jump, and also in the second case there is a latency in relation to the same simulation previously analyzed.

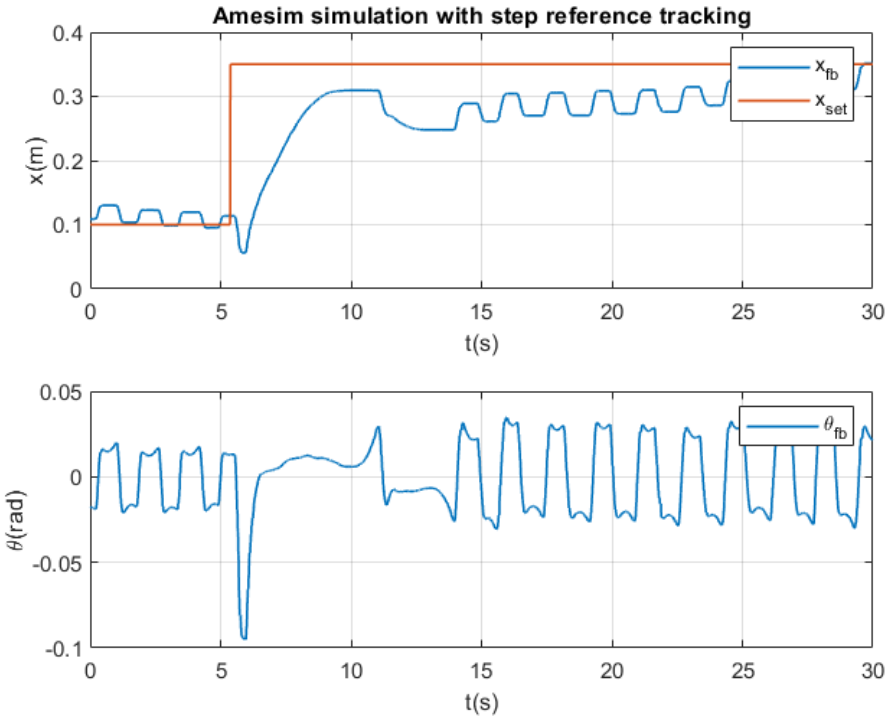


Figure 34: SiL Amesim simulation step constant reference tracking

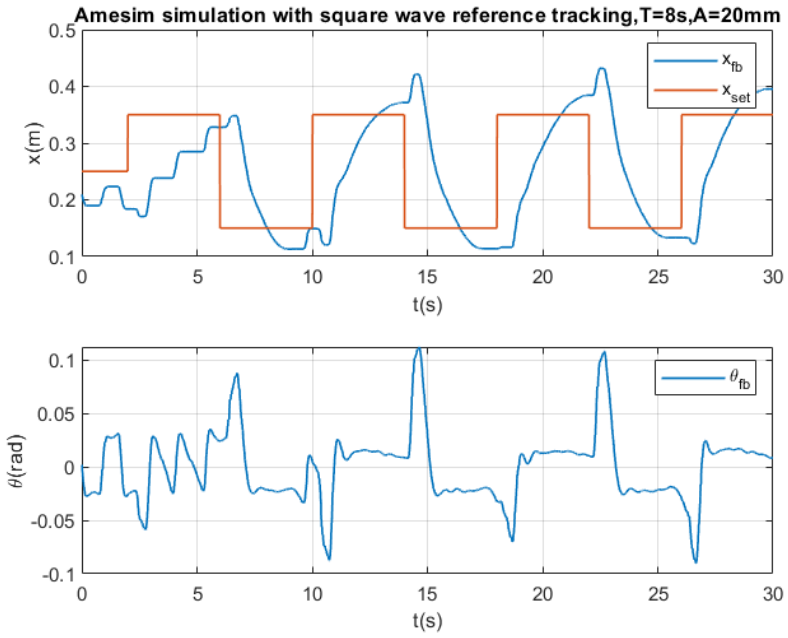


Figure 35: SiL Amesim simulation with square wave reference tracking, T=8s, A=200mm

5 Hardware-in-the-Loop: HiL

HiL simulation is the testing phase during which it is possible to validate a control algorithm running on an intended target controller, creating a virtual system that represents the physical system to control.

This is particularly interesting to:

- validate PLC control strategies online based on a virtualized controlled system yet able to represent the expected dynamics of the real machine,
- enhance or compare data measured from the field with simulated data (e.g. coming from virtual sensors),
- assist operators during real machine operation thanks to simulated predictions or diagnoses made by a "digital twin" fed with real data acquired from the field.

As already mentioned, one of the main potentialities of Amesim is to implement HiL, through the use of *Automation Connect*. In this case *Amesim* is used to simulate the behavior of a system controlled by a PLC, and through *Automation Connect* it is possible to continuously exchange data with a real PLC.

For this type of connection, a hardware communication gateway is required. We used a Siemens *SIMIT UNIT* hardware.

5.1 SIMIT UNIT

The hardware platform we used is the *SIMIT UNIT PN128*, which allows real-time simulations supporting up to 128 devices via *PROFINET*.

Using the *SIMIT UNIT,* the I/O signals of the decentralized periphery, as well as field device signals, such as actuators and sensors can be simulated.

Figure 36: SIMIT UNIT platform

The *SIMIT UNIT* replaces the field devices of a real plant. It simulates the behavior of these devices on the bus and communicates - like the real modules - with the automation hardware via the fieldbus.

5.2 Hardware configuration

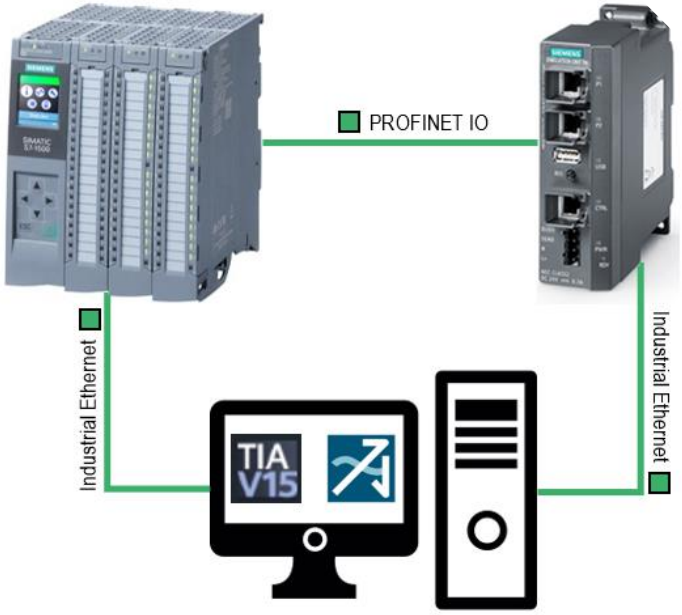


Figure 37: HiL hardware connection

The connection has been made in order to have our *CPU, S7-1500*, connected via *PROFINET* to the *SIMIT UNIT*. We then connected the two elements to our PC via Ethernet cable, so that the *SIMIT UNIT* could exchange data with *Amesim* and the PLC could be controlled via *TIA Portal*.

Differently from SiL, in this phase we had to add external modules to our PLC, because *SIMIT UNIT* only manages the I/O signals of the decentralized periphery, so to manage the input and output variables with *SIMIT UNIT* this operation was necessary, as shown in the Figure.

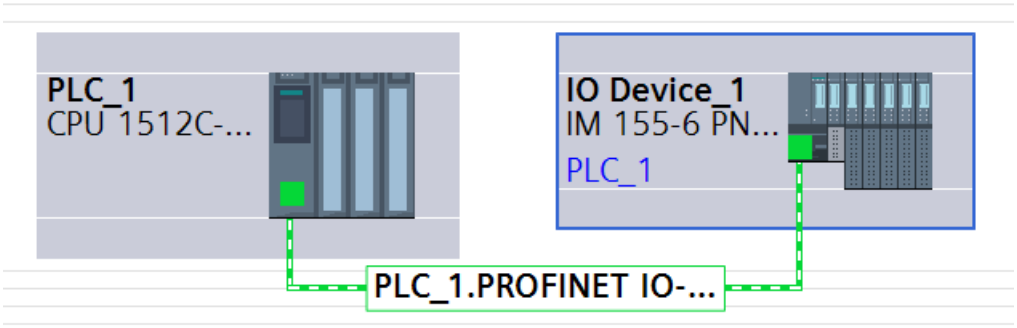


Figure 38: PLC hardware configuration

In order to use the external modules, it is necessary that the addresses of all the I/O variables are replaced, that is, those that will exchange data with *Amesim*, going to assign the addresses present on these modules, and it is also necessary to export the mapping, namely the allocation of addresses to the variables.

The other step was to load the hardware configuration on the *SIMIT UNIT*, using the appropriate software *SIMULATION UNIT 9.1*. Once done, the *SIMIT UNIT* recognizes which external devices we want to connect to our PLC and can virtualize them.

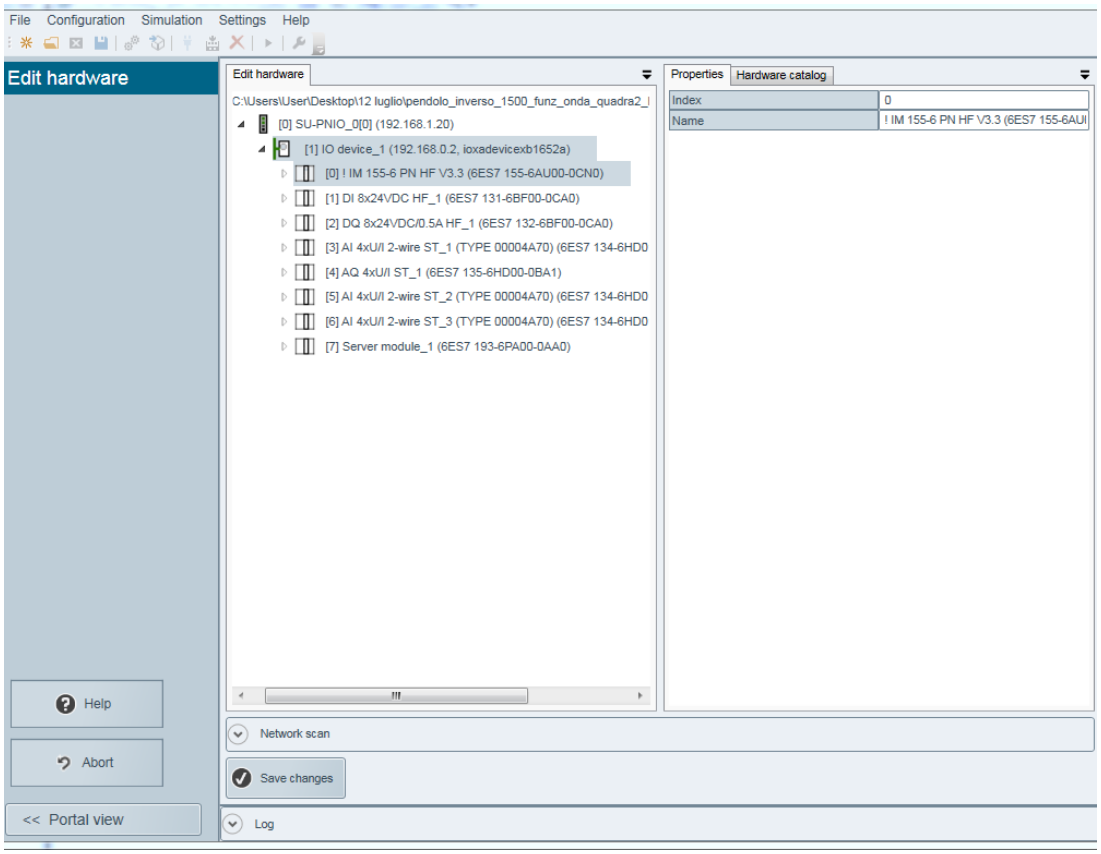


Figure 39: SIMIT UNIT software control panel

5.3 Software configuration

As already occurred for SiL, the tool that allows us to put *Amesim* in communication with *SIMIT UNIT* is *Automation Connect*, so choosing the relative tab it is possible to establish a connection between the latter and the hardware platform. In this case, only the addresses virtualized by *SIMIT UNIT* are loaded, that will be associated to the corresponding variables, importing the mapping that we created through *TIA Portal*.

The result is that in the following image: there are many empty spaces because not all the addresses on the external modules have been used.

As for the other simulations carried out with *Automation Connect*, we connected the variables of our interest in the *Variable Mapping tab*, and we started the simulation in the *Amesim* tab.

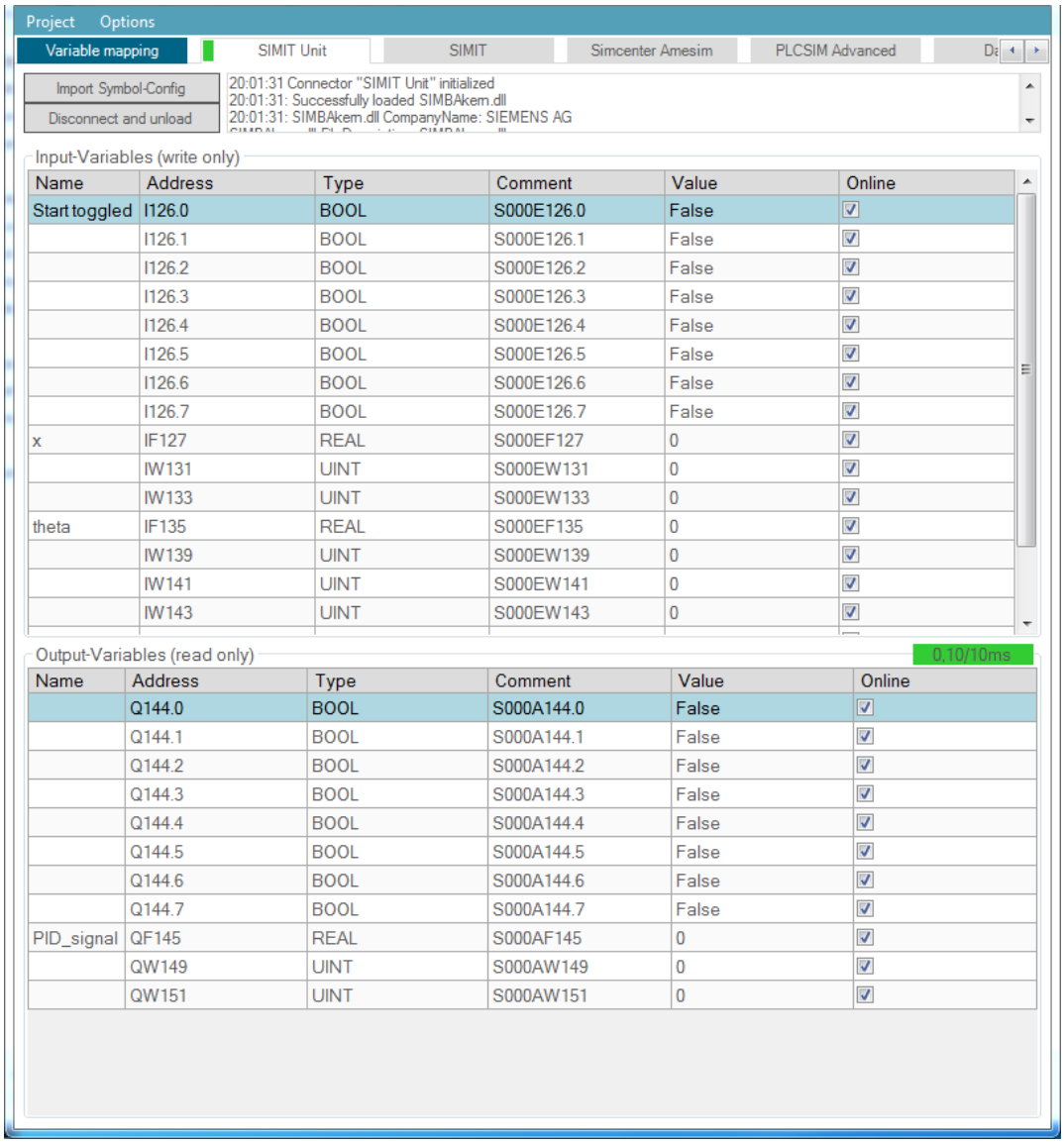


Figure 40: Automation Connect SIMIT UNIT tab

As for the *Amesim* model, the time-sync component of the type in Figure has been added and allows the synchronization of the simulation with the wall clock time. This element can be used when a model need to be piloted in "real time

conditions" from outside otherwise the data exchange with other real-time devices like PLCs will become very asynchronous. In fact, being the simulation in Amesim faster than the reality, it can be slowed down.

Figure 41: Time sync

5.4 Simulation results

Again, we could not use the same controllers as before, but it was necessary to make an optimization by obtaining the following controllers:

$$C_x = k_p + \frac{k_i}{s} + \frac{k_d s}{T_f s + 1} = 0{,}001 + \frac{0{,}001}{s} + \frac{0{,}08}{0{,}002s + 1}$$

$$C_\theta = k_p + \frac{k_i}{s} + \frac{k_d s}{T_f s + 1} = 5 + \frac{15{,}15}{s} + \frac{0{,}075}{0{,}144s + 1}$$

Also in this case, as in the SiL one, we noticed that the communication between the various components was affected by the very low target step. Certainly, the use of *SIMIT UNIT* helped to decrease the latency in the data transfer, so that the data obtained is very close to those obtained with the MiL and the real PLC.

Concerning the constant reference tracking, the system is stabilized even if it has some contained oscillations. Talking about the step reference tracking, we notice

a much lower rise time compared to the one obtained with the SiL and even after the jump the system has no problems to keep the reference.

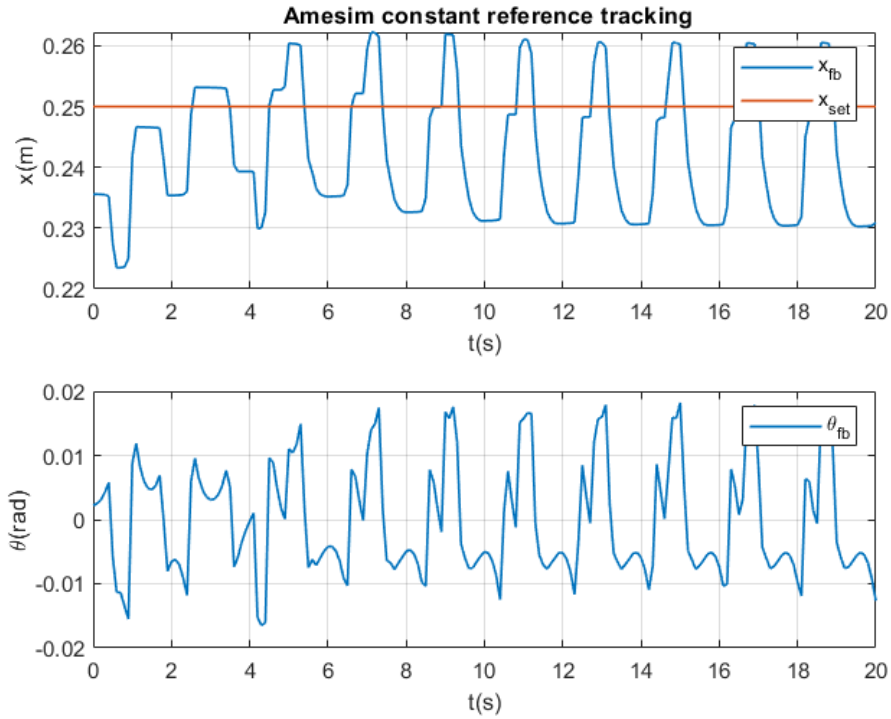


Figure 42: Amesim constant reference tracking

Finally, if we look at the last graph, we see how the system arrives a little late at the target point, so that once reached it does not have time to stabilize because the reference varies.

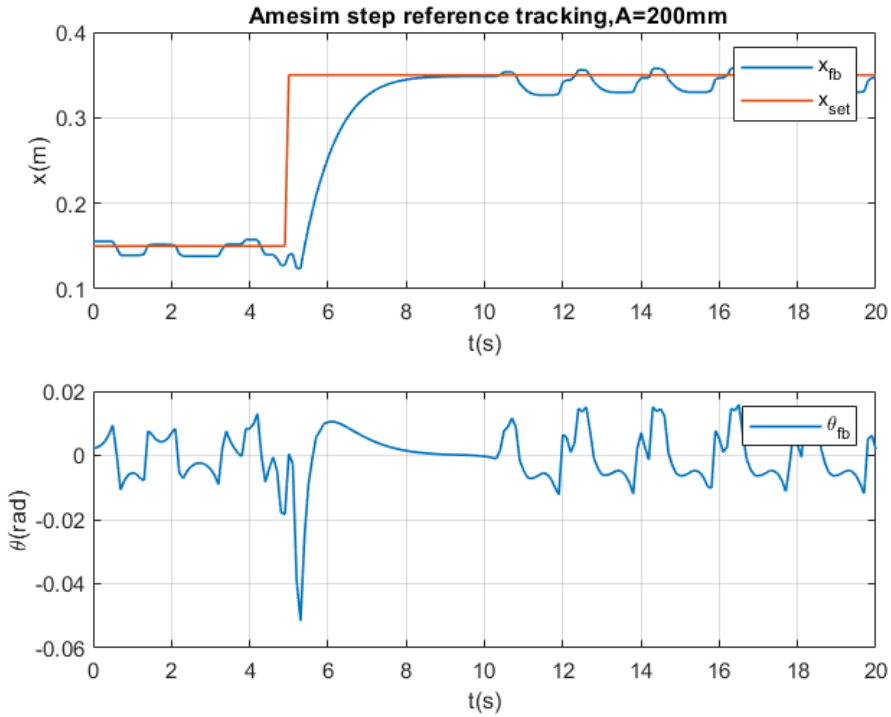


Figure 43: Amesim step reference tracking, A=200 mm

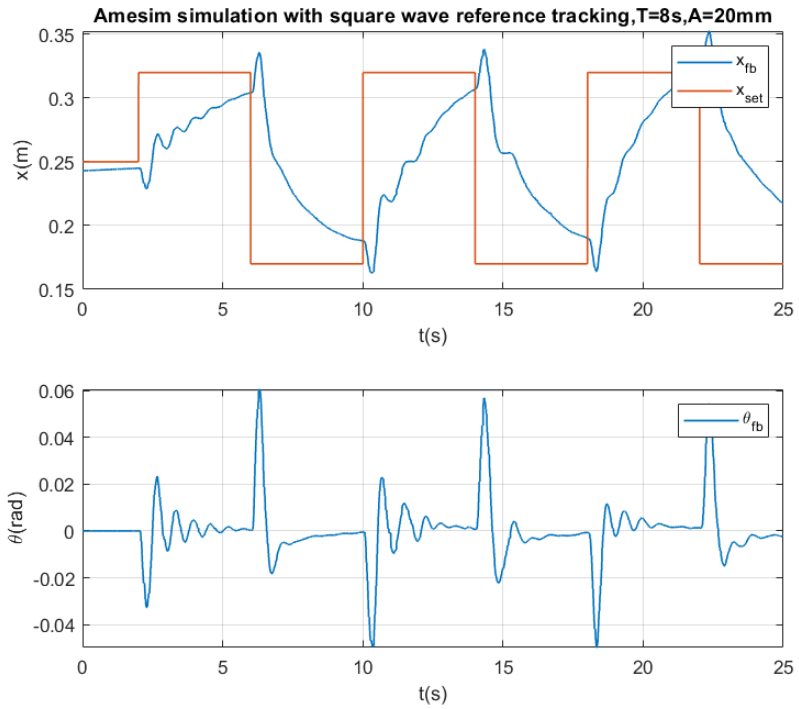


Figure 44: Amesim simulation with square wave reference, T=8s, A=200mm

6 Conclusions and future developments

The aim of this thesis was to create a reliable virtual model of a pneumatic system and, precisely, of a reverse pendulum.

From the results of the simulations it emerged that the various virtual models in the various phases of simulation presented a behavior very similar to that which they would have obtained through the tests carried out with the test bench.

To prove this, it was not necessary to substantially modify the controllers obtained in the various simulations, which proved to be effective even on the test bench.

The most significant differences were found in Software-in-the-Loop and Hardware-in-the-Loop: this was because with the means at our disposal, the communication between hardware and software was not able to withstand a data transfer fast enough for our type of system.

As for future developments, there is certainly still room for improvement regarding the reliability of the Amesim model: in fact, if it were possible to recover and make available precise data on friction in the piston, almost certainly it could be improved the behavior of the Amesim model making it more similar to the real one. These data were not at our immediate disposal and to obtain them were necessary long experimental tests that, for reasons of unsuitable instrumentation, it was not possible to perform.

Another improvement could be possible in case of a future update of the software we used, in order to obtain a more effective communication for systems like ours, which need a high speed of communication between the various actors involved.

As far as the stability of the inverse pendulum we have analyzed is concerned, it could be improved by using more complex functions than the PID compensators used in our system, which can guarantee better performance.

References

[1] Pontin M. (2018), *Modellazione, realizzazione e controllo mediante PLC di un sistema a pendolo inverso ad attuazione pneumatica*, Torino

[2] Petric, Josko & Situm, Zeljko. (2003). *Inverted Pendulum Driven by Pneumatics*, p. 19

[3] Krupke C. and Wang J. (2015), *Modelling and robust control of an inverted pendulum driven by a pneumatic cylinder*, IEEE International Conference on Advanced Intelligent Mechatronics (AIM), Busan, 2015, pp. 812-817. doi: 10.1109/AIM.2015.7222638

[4] Zilic, Tihomir & Pavkovic, Danijel & Zorc, Davor. (2009). *Modeling and control of a pneumatically actuated inverted pendulum. ISA transactions*. 48. 327-35. 10.1016/j.isatra.2009.03.004.

Siemens:

https://new.siemens.com/global/en.html, last access 12th January 2019

Simcenter Amesim:

https://www.plm.automation.siemens.com/global/it/products/simcenter/simcenter-amesim.html, last access 23rd March 2019

Simit Unit:

https://new.siemens.com/global/en/products/automation/industry-software/simit.html, last access 19th February 2019

Simulink:

https://it.mathworks.com/products/simulink.html, last access 24th February 2019

Tia Portal:

https://w3.siemens.com/mcms/automation-software/en/tia-portal-software/step7-tia-portal/pages/default.aspx, last access 15th March 2019

Appendix A

A.1. Matlab scripts

In the following page some scripts used in Matlab are showed. They were used for the friction parameter approximation.

A.1.1. Main

```
load("C:\Users\Paolo\Desktop\Tesi\Amesim\controllo posizione simucosim\data1.mat");

Par.Data.dt_max=min(diff(t));
Par.Data.t=t(5376:9334)
Par.Data.x=[Par.Data.t, x(5376:9334)];
Par.Model.dynamic_friction_pressure_grad=8;
Par.Model.dynamic_to_stiction_friction_co=1.4;
Par.Model.stick_displacement_treshold=0.1;
Par.Model.equivalent_viscous_friction_dur=3000;
Par.Model.mass=0.7;
Par.Model.fric0=4;
Par.Model.area=1.5;
Par.Model.cq=0.4;
Par.Model.coefv=5;

x_try=[8 1.4 0.1 3000 0.7 4 1.5 0.4 5];
x_opt=fminsearch(@(x)opt_par(Par ,x), x_try)
```

A.1.2. Parameters optimization

```
function [e] = opt_par(Par,x)
%UNTITLED3 Summary of this function goes here
%   Detailed explanation goes here
Par.Model.dynamic_friction_pressure_grad=x(1);
Par.Model.dynamic_to_stiction_friction_co=x(2);
Par.Model.stick_displacement_treshold=x(3);
Par.Model.equivalent_viscous_friction_dur=x(4);
Par.Model.mass=x(5);
Par.Model.fric0=x(6);
Par.Model.area=x(7);
Par.Model.cq=x(8);
Par.Model.coefv=x(9);
Parr=[Par.Model.coefv; Par.Model.fric0; Par.Model.area; Par.Model.cq;...
    Par.Model.dynamic_friction_pressure_grad];

% Lancio il modello
opz = simset('SrcWorkspace','current','DstWorkspace','current');
sim('controllo_posizione_simucosim1.slx',[],opz);
e=sum((Y(5:end,3)-Y(5:end,2)).^2);
display(e);
plot(Y(5:end,2));
hold on
plot(Y(5:end,3))
grid on
hold off
end
```

Appendix B

B.1. Source code of program for *PLC S7-1500*

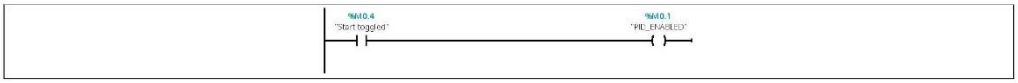
In the following pages, the source code is showed and written through the software *Tia Portal*. It is uses on *PLC S7-1500.*

Main [OB1]

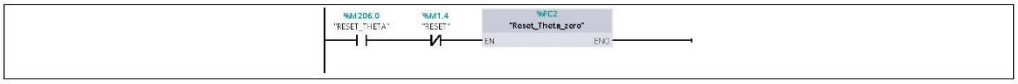
Main Proprietà							
Generale							
Nome	Main	**Numero**	1	**Tipo**	OB	**Linguaggio**	KOP
Numerazione	Automatico						
Informazioni							
Titolo	"Main Program Sweep (Cycle)"	**Autore**		**Commento**		**Famiglia**	
Versione	0.1	**ID definito dall'utente**					

Nome	Tipo di dati	Valore di default	Commento
Input			
Initial_Call	Bool		Initial call of this OB
Remanence	Bool		=True, if remanent data are available
Temp			
Constant			

Segmento 1: Amesim start toggled

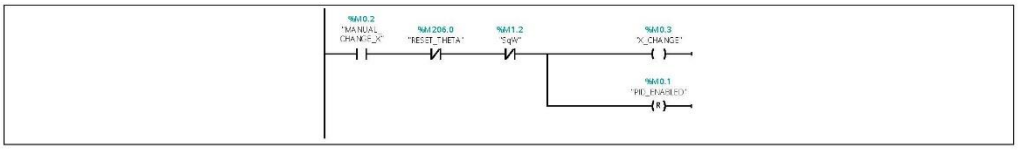


Segmento 2: Start reset theta zero

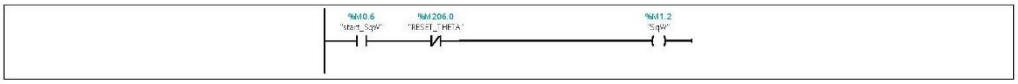


Segmento 3: Start update PID parameters

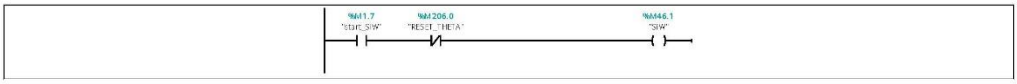
Segmento 4: Change position of cart



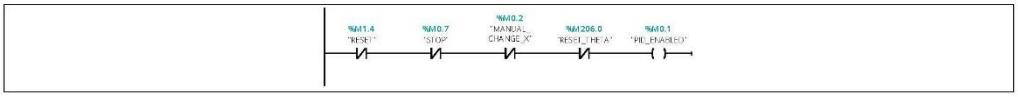
Segmento 5: Start square wave



Segmento 6: Start sine wave



Segmento 7: Stop program



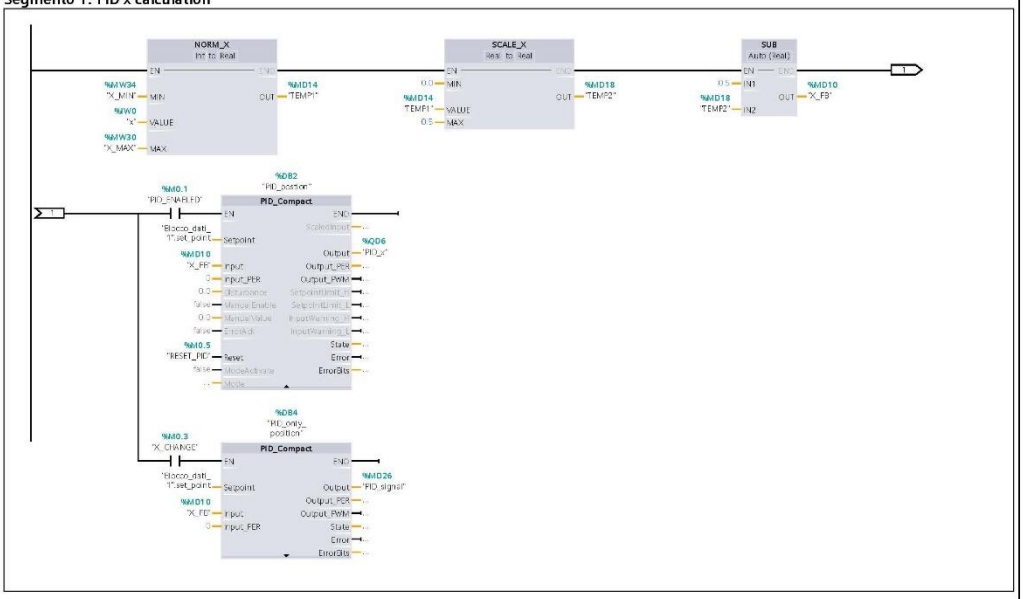
PID Cyclic interrupt [OB30]

PID Cyclic interrupt Proprietà							
Generale							
Nome	PID Cyclic interrupt	**Numero**	30	**Tipo**	OB	**Linguaggio**	KOP
Numerazione	Automatico						
Informazioni							
Titolo		**Autore**		**Commento**		**Famiglia**	
Versione	0.1	**ID definito dall'utente**					

Nome	Tipo di dati	Valore di default	Commento
Input			
Initial_Call	Bool		Initial call of this OB
Event_Count	Int		Events discarded
Temp			
Constant			
set_point	Real	0.25	
THETA_ZERO	Real	0.011	

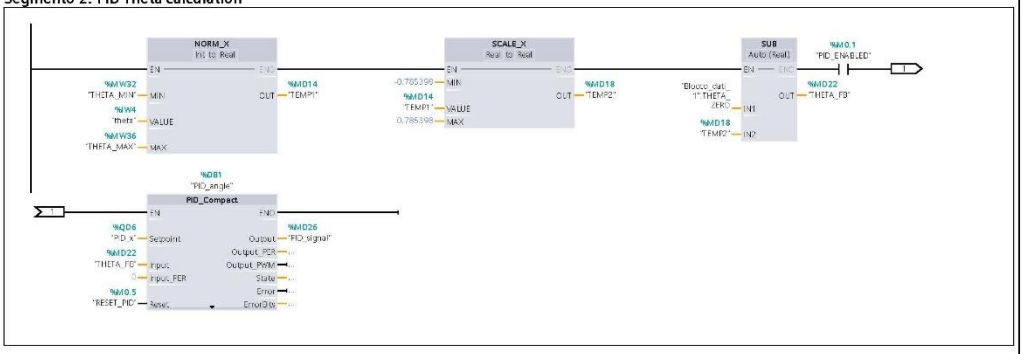
Segmento 1: PID x calculation

Segmento 1: PID x calculation

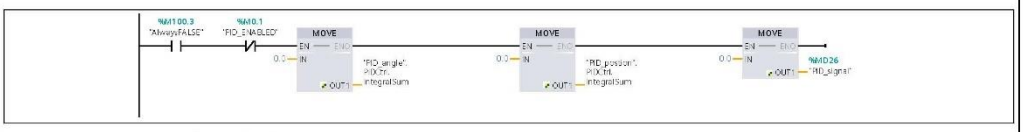


Segmento 2: PID Theta calculation

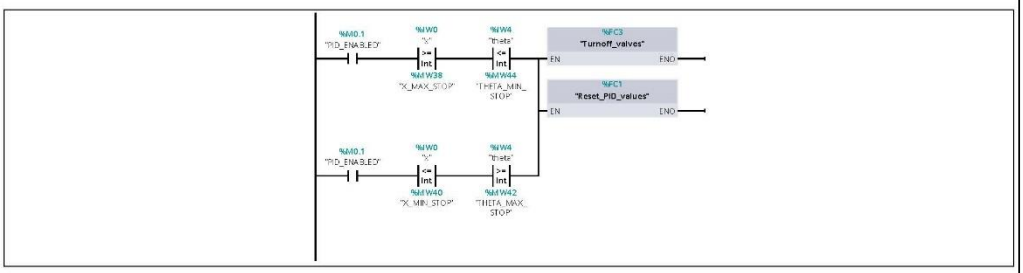
Segmento 2: PID Theta calculation



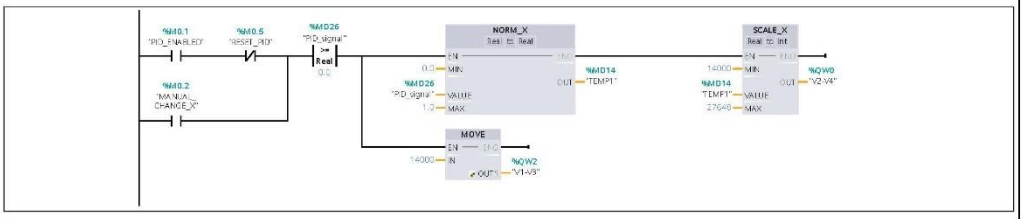
Segmento 3:



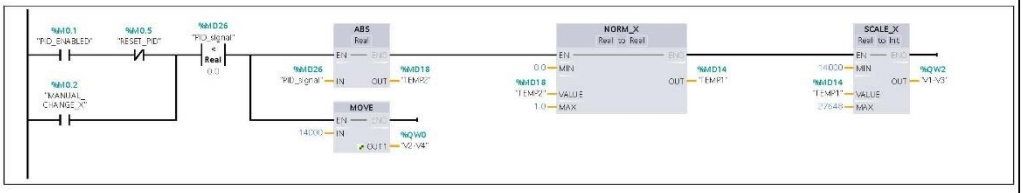
Segmento 4: Reset PIDs and turn off valves



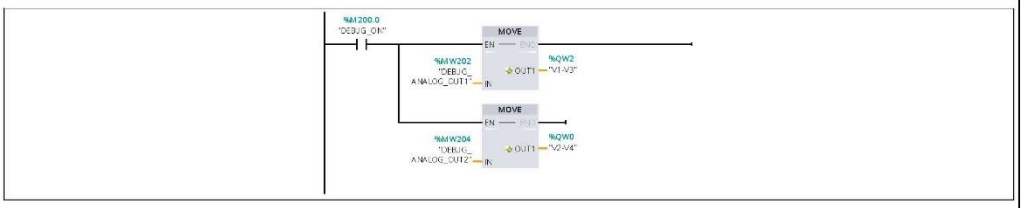
Segmento 5: if PID_signal>0 power V2-V4



Segmento 6: if PID_signal<0 power V1-V3



Segmento 7:


%M200.0
"DEBUG_ON"
MOVE
EN
ENO
%MW202
"DEBUG_
ANALOG_OUT1"
IN
OUT1
%QW2
"V1-V3"
MOVE
EN
ENO
%MW204
"DEBUG_
ANALOG_OUT2"
IN
OUT1
%QW0
"V2-V4"

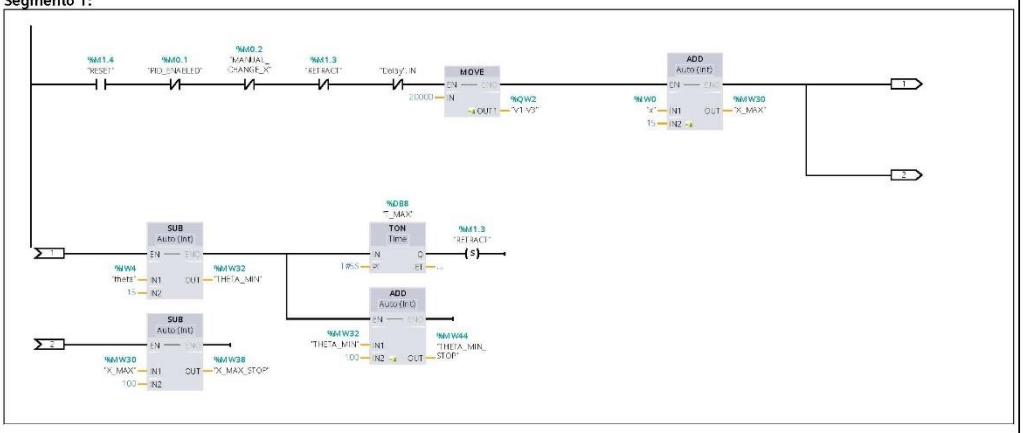
Reset_x_theta_min_max [OB124]

Reset_x_theta_min_max Proprietà							
Generale							
Nome	Reset_x_theta_min_max	**Numero**	124	**Tipo**	OB	**Linguaggio**	KOP
Numerazione	Automatico						
Informazioni							
Titolo	"Main Program Sweep (Cycle)"	**Autore**		**Commento**		**Famiglia**	
Versione	0.1	**ID definito dall'utente**					

Nome	Tipo di dati	Valore di default	Commento
Input			
Initial_Call	Bool		Initial call of this OB
Remanence	Bool		=True, if remanent data are available
Temp			
Constant			

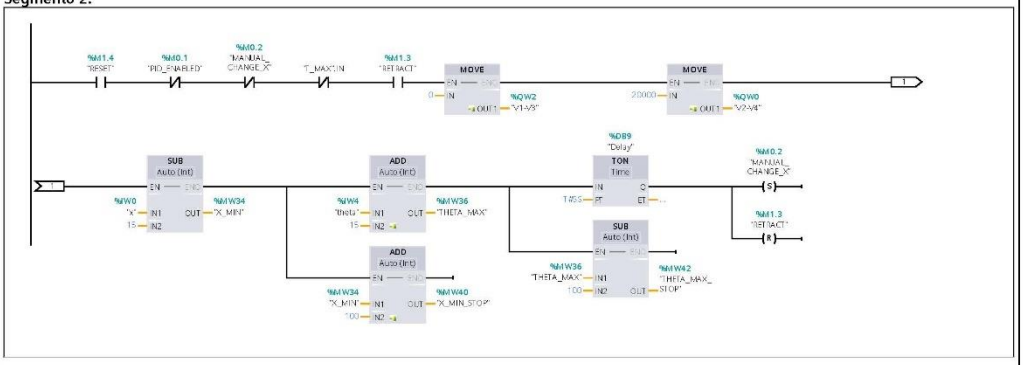
Segmento 1:

Segmento 1:

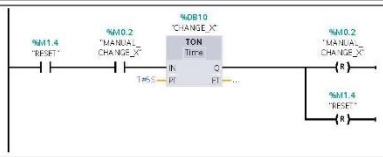


Segmento 2:

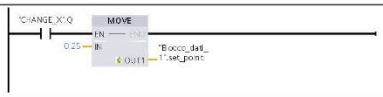
Segmento 2:



Segmento 3:


%DB10
"CHANGE_X"
%M1.4
"RESET"
%M0.2
"MANUAL_
CHANGE_X"
TON
Time
IN
Q
PT
%M0.2
"MANUAL_
CHANGE_X"
%M1.4
"RESET"

Segmento 4:

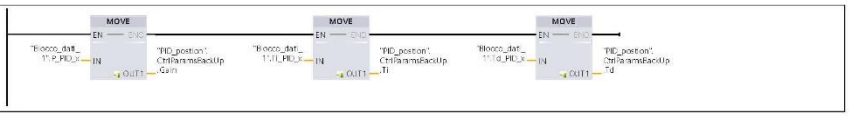

"CHANGE_X".Q
MOVE
EN
ENO
0.25
IN
OUT1
"Blocco_dati_
1".set_point

Update_PID_parameters [FC6]

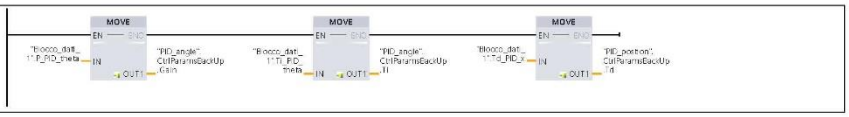
Update_PID_parameters Proprietà							
Generale							
Nome	Update_PID_parameters	**Numero**	6	**Tipo**	FC	**Linguaggio**	KOP
Numerazione	Automatico						
Informazioni							
Titolo		**Autore**		**Commento**		**Famiglia**	
Versione	0.1	**ID definito dall'utente**					

Nome	Tipo di dati	Valore di default	Commento
Input			
Output			
InOut			
Temp			
Constant			
▼ Return			
Update_PID_parameters	Void		

Segmento 1:



Segmento 2:



Turnoff_valves [FC3]

Turnoff_valves Proprietà							
Generale							
Nome	Turnoff_valves	**Numero**	3	**Tipo**	FC	**Linguaggio**	KOP
Numerazione	Automatico						
Informazioni							
Titolo		**Autore**		**Commento**		**Famiglia**	
Versione	0.1	**ID definito dall'utente**					

Nome	Tipo di dati	Valore di default	Commento
Input			
Output			
InOut			
Temp			
Constant			
▼ Return			
Turnoff_valves	Void		

Segmento 1:

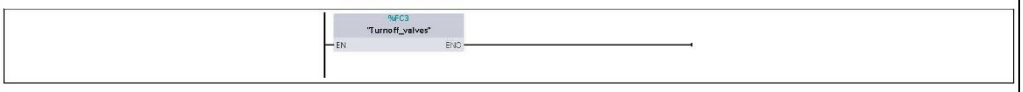
turnoff V1-V2-V3-V4

Reset_Theta_zero [FC2]

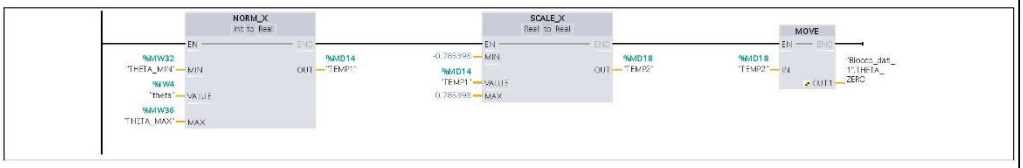
Reset_Theta_zero Proprietà							
Generale							
Nome	Reset_Theta_zero	**Numero**	2	**Tipo**	FC	**Linguaggio**	KOP
Numerazione	Automatico						
Informazioni							
Titolo		**Autore**		**Commento**		**Famiglia**	
Versione	0.1	**ID definito dall'utente**					

Nome	Tipo di dati	Valore di default	Commento
Input			
Output			
InOut			
Temp			
Constant			
▼ Return			
Reset_Theta_zero	Void		

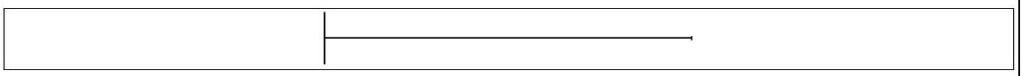
Segmento 1: Turn off valves



Segmento 2: set offset theta



Segmento 3:

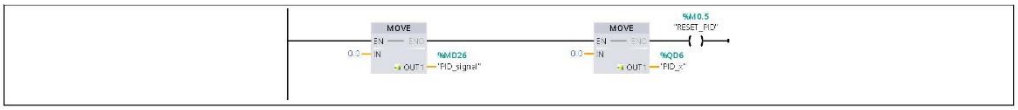


Reset_PID_values [FC1]

Reset_PID_values Proprietà							
Generale							
Nome	Reset_PID_values	**Numero**	1	**Tipo**	FC	**Linguaggio**	KOP
Numerazione	Automatico						
Informazioni							
Titolo		**Autore**		**Commento**		**Famiglia**	
Versione	0.1	**ID definito dall'utente**					

Nome	Tipo di dati	Valore di default	Commento
Input			
Output			
InOut			
Temp			
Constant			
▼ Return			
Reset_PID_values	Void		

Segmento 1: Reset PIDs

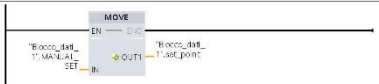


Change_setpoint_x [FC5]

Change_setpoint_x Proprietà							
Generale							
Nome	Change_setpoint_x	**Numero**	5	**Tipo**	FC	**Linguaggio**	KOP
Numerazione	Automatico						
Informazioni							
Titolo		**Autore**		**Commento**		**Famiglia**	
Versione	0.1	**ID definito dall'utente**					

Nome	Tipo di dati	Valore di default	Commento
Input			
Output			
InOut			
Temp			
Constant			
Return			
Change_setpoint_x	Void		

Segmento 1: Change setpoint



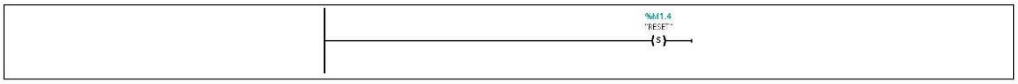
Startup [OB100]

Startup Proprietà							
Generale							
Nome	Startup	**Numero**	100	**Tipo**	OB	**Linguaggio**	KOP
Numerazione	Automatico						
Informazioni							
Titolo	"Complete Restart"	**Autore**		**Commento**		**Famiglia**	
Versione	0.1	**ID definito dall'utente**					

Nome	Tipo di dati	Valore di default	Commento
Input			
LostRetentive	Bool		True if retentive data are lost
LostRTC	Bool		True if date and time are lost
Temp			
Constant			

Segmento 1:

Segmento 2:

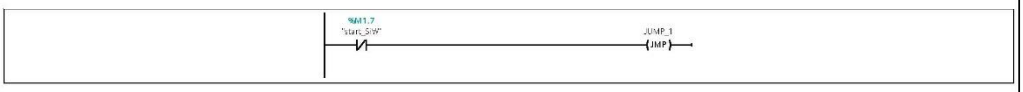


Sine_Wave_generator [OB31]

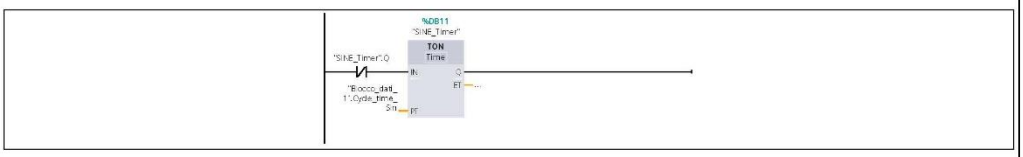
Sine_Wave_generator Proprietà							
Generale							
Nome	Sine_Wave_generator	**Numero**	31	**Tipo**	OB	**Linguaggio**	KOP
Numerazione	Automatico						
Informazioni							
Titolo		**Autore**		**Commento**		**Famiglia**	
Versione	0.1	**ID definito dall'utente**					

Nome	Tipo di dati	Valore di default	Commento
Input			
Initial_Call	Bool		Initial call of this OB
Event_Count	Int		Events discarded
Temp			
Constant			

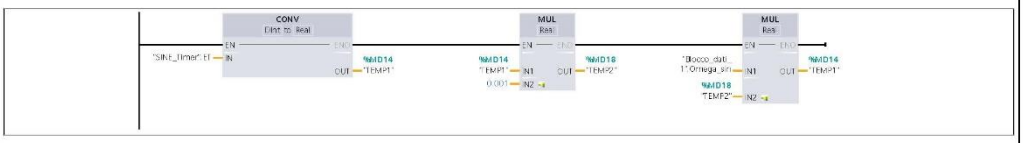
Segmento 1: BIT check



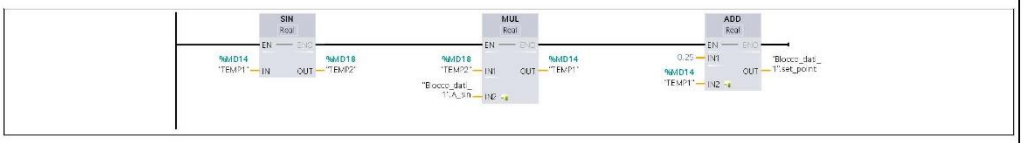
Segmento 2: start Timer



Segmento 3: omega*t



Segmento 4: A*sin(omega*t) and update set_point



Segmento 5: reset setpoint