

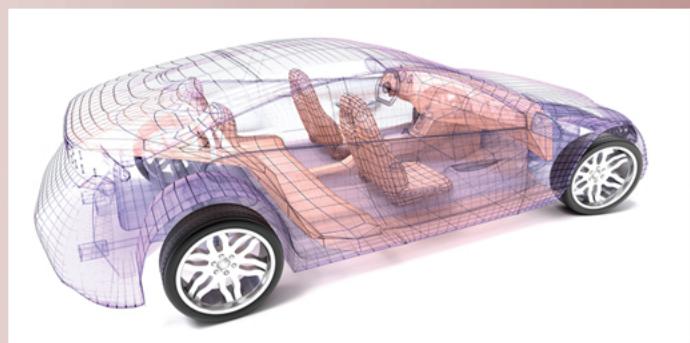
Modeling and Simulation of Multi-Physics Systems with MATLAB – Simulink for Students and Engineers

Second edition

Introduction to Model-Based-Design



- *MATLAB*
- *Simulink*
- *Simscape*
- *Stateflow*



Ivan LIEBGOTT

Modeling and Simulation of Multi-Physics Systems with MATLAB-Simulink for students and engineers

Introduction to Model-Based-Design

Author: Ivan LIEBGOTT

*Professor of Preparatory Classes for Postgraduate Schools
University of Nice (France)*

*Engineer with a degree from the "Institut National Supérieur des Sciences Appliquées" (INSA)
MASTER in the design of Aeronautic and Spatial structures
Professor of Engineering Sciences*

ivan.liebgott@gmail.com

Follow me on  LinkedIn

This book was created to be freely shared with the community of MATLAB users.
Your comments and suggestions are welcome and will help me to progress and improve this work.



This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

Ivan LIEBGOTT 2016

Preface:

Engineers are at the heart of the design process of complex systems and must meet the challenges of competitiveness, innovation and performance on a daily basis. This cannot be done without integrating structured industrial processes, nor without mastery of modern modeling and simulation tools. At each stage of the design cycle, the methods used must enable the lowering of costs, a reduction in the risk of error and a minimizing of the impact of errors.

At the heart of this process, digital modeling and simulation play a major role in allowing engineers to anticipate, understand and verify the analyses that they conduct throughout a project.

Standard industrial procedures, such as the V-Model, fully integrate digital simulation through associated methods such as "Model Based Design". Modern simulation tools facilitate the creation of complex global models integrating all of the components of a system and taking all interactions into account. This procedure is called multi-physics modeling. The real system may be advantageously replaced by its digital model in order to conduct test that mobilize significant human and material resources ahead of time. This procedure requires the availability of validated models that faithfully reproduce the behavior of real systems.

This book will provide you with a multi-physics modeling approach that uses the functionalities and innovations of simulation software in order to make the modeling process quicker and more efficient. The simulation platform used is MATLAB/Simulink software, release 2015b.

The work is intended to provide the keys to facilitate the global modeling of a system by creating the link between industrial methods and methods used in the engineer training cycle. This is illustrated by numerous examples in different technological domains (electric, hydraulic, mechanical...) and highlights the interconnection of physics domains.

The fundamentals for all of the tools necessary to conduct this procedure are presented:

- MATLAB
- Simulink
- Simscape
- SimHydraulics
- SimMechanics
- SimElectronics
- Stateflow

This work suggests an introduction to their use and will not make you an expert in any of them. You may, on the other hand, use it to perceive their full potential, and exploit it in greater depth in accordance with the specific needs that you encounter in your modeling.

Happy reading,

Ivan Liebgott

Table of Contents

Chapter 1: Concepts and Strategies in Modeling

I. Introduction.....	10
II. Industrialization and design cycle of a system.....	11
A. Competences of the engineer.....	12
B. The performance triptyc(*)	13
III. Using the procedure - Introduction to “Model Based Design”	14
A. Material architecture of the project.....	14
B. Analysis and Requirements Phase.....	15
C. Design-Modeling-Simulation Phase	16
1. “White box” modeling	17
2. “Multi-physics” modeling	19
3. Model simulation.....	20
4. Comparison of simulated and measured performances.....	21
5. “Gray box” modeling	22
6. Model-in-the-loop (MIL)	25
D. Coding Implementation Phase.....	26
1. Software-in-the-Loop (SIL).....	27
2. Processor-in-the-loop (PIL).....	27
E. Integration and Testing Phase.....	28
1. Hardware-In-the-Loop (HIL)	28
F. Validation Receipt Phase.....	31

Chapter 2: Introduction and presentation of modeling tools

I. MATLAB-Simulink software.....	32
A. Description and hierarchy of tools used.....	32
1. MATLAB.....	32
2. Simulink.....	33
3. Simscape.....	34
4. Stateflow	37
5. Using modeling tools	37
II. Presentation of the MATLAB – Simulink environment	39
A. Launching the software	39
B. The MATLAB environment window	39
1. The MATLAB command bar	40
C. The Simulink environment window	40
D. MATLAB – Simulink Configuration	42
1. Naming a file in MATLAB/Simulink.....	42
2. The MATLAB “path”	42
3. Adding files to the “path” for all sessions	42

4. Add files to the “path” for the current session	43
III. Design strategy for a Multi-Physics Model	44
A. Link to the Energy chain/information chain diagram	44
IV. Application to a hydraulic boat pilot	46
A. Diagram showing the energy chain and information chain of the boat hydraulic pilot	47
B. Multi-physics model of the boat hydraulic pilot executed with MATLAB - Simulink	48
C. Model loading and simulation	49
D. Visualization of the results of the multi-physics model.....	49
E. Exploring the model.....	55
1. Exploring the information chain model: Simulink and Stateflow	55
2. Exploring the energy chain model: Simscape Electric Library	56
3. Exploring the energy chain: SimHydraulics	57
4. Exploring the energy chain: SimMechanics 2G.....	58
5. Exploring the energy chain: Simulink	59
F. Interactive piloting of the model	59
V. Examples of multi-physical models and possible uses	62
A. The Maxpid robot.....	62
B. The linear axis Control'X.....	67
C. How to make a multi-physical model with MATLAB-Simulink.....	71

Chapter 3: Getting to grips with Simscape

I. Introduction to acausal modeling with Simscape.....	73
A. Choice of components	74
B. Placement and assembly of components.....	75
C. Different types of ports and connections.....	76
D. Configuration of components	78
E. Launching the simulation and analyzing the results.....	81
II. Comparison with the causal approach.....	83
A. Equation for system behavior	83
B. Choice of components	83
C. Placement and assembly of components	83
D. Configuration of components	84
E. Launching the simulation and analyzing the results	85
F. Advantages and disadvantages of the causal and acausal approaches	87
III. Modeling fundamentals with Simscape	88
A. Ideas of physical fields	88
B. Important blocks in Simscape	89
C. “Across” and “Through” type variables and position sensors	90
D. Orientation of components.....	90
1. The use of active components	91
2. Placing and orienting the sensors.....	94
3. Use of components with an oriented dynamic.....	97
4. Use of passive components.....	97
5. Choice of solver	97

6. Problems which the solver may encounter	98
IV. Examples of multi-field modeling.....	99
A. Electro-mechanical field - Linear axis.....	99
1. Choice of components	100
2. Placement and assembly of components.....	102
3. Configuration of components	103
4. Open loop model simulation	111
5. Using the Simscape Data-logger	112
6. Creation of sub-systems.....	116
7. Modeling the servo position of the axis.....	123
B. Hydraulic-mechanical fields - single-acting hydraulic cylinder	128
1. Choice of components	129
1. Placement and assembly of components.....	131
2. Configuration of components	132
3. Simulation.....	137
4. Using the signal routing functionalities	138
5. Replacing the pressure source with a flow source.....	141
C. Electric field - PWM control of a DC motor	144
1. Use of the “Controlled PWM Voltage” component.....	145
2. PWM control for a DC motor.....	148
3. Using of the “H-Bridge” component.....	150
D. Rendering an interactive model, using the “dashboard” library	156
1. Example of an interactive model.....	158
2. Using the library blocks.....	159
V. Teaching applications	164
A. Introduction to the buck converter.....	164
B. Teaching objectives.....	166
C. Constructing the model.....	168
D. The instructionalization of the model.....	176
1. Creating a sub-system and adding an image.....	176
2. Instrumentation of the model.....	176
3. Conclusion on rendering a model suitable for teaching	180
4. Optimize the instructionalization of the model based on the learning objective.....	182
E. Using the results from the model simulation.....	186
1. Objective 1: Understand the circulation of the current in the circuit in the active and freewheel phases.	186
2. Objective 2: Visualize and evaluate the effect of the duty cycle on the motor current.....	188
3. Objective 3: Visualize and evaluate the effect of the chopping frequency on the current ripple	190
4. Objective 4: Visualize and evaluate the effect of the load inductance on the current ripple	192
F. Conclusion.....	194

Chapter 4: MATLAB management

I. Introduction.....	195
A. Creating a variable.....	195

B. Creating a vector.....	196
C. Indexing vector components.....	196
D. Plotting curves.....	197
E. Elementary curve shaping.....	199
F. Graphics annotation.....	201
G. Creating an elementary script.....	203
H. MATLAB comparison operators.....	206
I. Structure of normal loops	206
1. Syntax for the if – elseif – else loop	206
2. Loop syntax for.....	206
3. Syntax of the while loop	207
II. Example of uses.....	207
A. Interpolating a series of data	207
B. Symbolic calculation with MATLAB.....	209
1. Solving an algebraic equation.....	209
2. Expanding or factoring an expression	210
3. Deriving a function	211
4. Integrating a function.....	212
5. Using Laplace transforms	212
6. Using inverse Laplace transform.....	213
7. Partial fraction decomposition.....	215
1. Solving a first order differential equation.....	216
1. Solving a second order differential equation.....	217
C. Manipulating transfer functions	220
1. Creating a transfer function.....	220
2. Transfer function operations.....	221
3. Tracing the time responses of a system	224
4. Tracing frequency responses of a system	225
5. Evaluate gain and phase margins.....	227
6. Summary table of useful commands for transfer functions	228

Chapter 5: Simulink management

I. Introduction.....	230
II. Controlling the temperature of an oven.....	230
A. Opening the model.....	231
B. Opening the script containing the definition of the variables.....	232
C. Launching the simulation	233
D. Plotting a Bode diagram with Simulink	234
1. Plotting an open loop Bode diagram	235
2. Plotting a closed loop Bode diagram	238
E. Plotting a Black-Nichols diagram	242
F. Adding and configuring a saturation	242
G. Exporting the simulation variables to the Workspace	245
1. Writing a script to plot a series of curves.....	248

I. Introduction to Stateflow	250
------------------------------------	-----

Chapter 6: Stateflow management

A. Modelling a state machine reference course with Stateflow	250
B. Creating a state diagram	250
1. Opening the model.....	250
2. Inserting a “chart”	251
C. Creating an elementary state diagram.....	252
1. Creating states	252
2. Creating a default transition	253
3. Creating transitions.....	253
4. Creating actions in states.....	253
5. Creating transition labels	254
6. Definitions of input and output variables of the state diagram.....	255
7. Simulating a states diagram	258
D. Architecture of state machines	259
1. States hierarchy	259
2. Test priorities for transitions	259
3. Parallel states.....	260
E. Adding hierarchical levels and parallel states into a state diagram	260
F. Summary and additional useful Stateflow commands.....	266

Chapter 7: SimMechanics management

I. Introduction to SimMechanics	268
A. Analyzing a SimMechanics 2G model	268
B. Configuring gravity.....	270
II. Integrating a SimMechanics model into a multi-physics model.....	271
A. Model connections.....	272
B. Interfacing between Simscape and SimMechanics.....	273
1. Translational interface between Simscape and SimMechanics	274
2. Rotational interface between Simscape and SimMechanics	274
3. Adding ports to a connection.....	276
4. Modeling a variable external force	280
C. Results of the simulation.....	283
III. Importing a SolidWorks model into SimMechanics	284
A. Principles	284
B. Installing “SimMechanics Link”	284
C. Converting a Solidworks assembly file to xml file	286

Chapter 8: Identifying a model

I. Black-box modeling, identification.....	291
A. The model.....	291
B. Implementing the method using the Identification toolbox	292
1. Analysis of the data used for identification	292
2. Opening and presentation of the “System Identification” toolbox.....	293
3. Importing data.....	294
C. Using the command line method.....	299

Chapter 9: Control command with MATLAB - Simulink

I. Introduction.....	302
II. Automatic PID settings.....	302
A. Modeling.....	302
B. Opening the model.....	303
1. Analysis of the time domain response.....	305
2. Importing into Simulink.....	309
III. Manual adjustments to a PID with the “compensator design” tool	310
A. Opening the model.....	310
B. Adjusting the PID.....	311
1. Placement of linearization points	311
2. Choosing a block to adjust.....	313
3. Choosing plots to display for the open-loop	314
4. Choosing plots to view the performance of the closed-loop.....	315
5. Analyze the graphic windows of the “Control System Designer” window	317
6. Adjusting the PID.....	322
7. Definition and display of performance criteria	322
8. Adjusting the PID with the help of cursors	325
9. Exporting settings in the Simulink model	329
IV. Designing and setting a compensator of any form.....	330
A. Opening the model.....	330
B. Designing a compensator	331
1. Choosing a block to adjust.....	332
2. Choosing plots to display for the open-loop	333
3. Choosing plots to view the performance of the closed-loop.....	334
4. Analyze the graphic windows of the “compensator design” tool	335
5. Viewing the gain influence of the closed-loop transfer function	336
6. Adding an integrator.....	340
7. Adding a lead compensator (Lead)	341
8. Adding a Notch	344
9. Adjusting a Notch	346
10. Exporting the compensator transfer function to the Simulink model	350

Chapter 1: Concepts and Strategies in Modeling

I. Introduction

The rapid evolution of technologies pushes man to design systems that are ever more complex, the behavior of which can no longer be modeled without the use of modeling software.

Since the interactions between the components are numerous, it is of primary importance to be able to model the system in its entirety by taking its structure, its control system and its environment into consideration. Energy and information chains can no longer be modeled separately, but must be integrated into a single model. Using this model, it is possible to anticipate the behavior of the real system in the design phase in order to enable the real system to perform its function.

Modeling of complex systems is at the heart of engineer training. This approach, which is based on breaking complexities, shows the sub-systems that drive each of the different technologies. Departitioning technology universes is therefore necessary for implementing a relevant modeling procedure. The rapid evolution of simulation software now allows us to model all of the parts of a system using a single program. This multi-disciplinary approach facilitates the management of exchanges between sub-systems and the availability of results which take into consideration all of the physical phenomena that enter into the functioning of a system. The relevance of results is increased and it becomes possible to examine interactions. In contrast, the mastering of simulation tools drives a new, broader field of competences, which requires a structured learning process. This new approach requires a tool for modeling and simulating a unique, visual and interactive environment, industrially validated and offering all the modeling tools for the information chain (conventional control, state diagrams ...) and energy chain (advanced component libraries, integrated CAD model...).

In order to make the training procedure effective, the methods and concepts used in the training cycle for future engineers must be the same as in the industrial world. We will first see how it is possible to connect industrial design standards with the concepts currently used in engineer training courses.

The procedure used in industry by teams of engineers to design and develop a system is a complex process that most often complies with industrial standards such as the "V-Model" and associated methods such as "Model Based Design". These procedures require the availability of validated models that teams of engineers will use to replace real material in the design and adjustment phases. "Modeling" skill is therefore at the heart of the learning process and remains one of the basic competences of an engineer. The development of modeling and simulation software has radically modified the methods of future engineers for acquiring this competence. The modeling process is no longer limited to writing differential equations which govern the behavior of a system and attempt to solve them analytically in the rare cases where this is possible. Relevant use of modeling and simulation software allows the use of different approaches to quickly model and obtain valid results:

- *"White box" modeling:*

This approach consists of writing equations and solving them using a digital tool. This obtains an initial approximation of the model structure. Generally, the parameters that characterize the behavior of the system are not yet completely known. They must then be ignored or estimated to conduct the simulation. The results obtained must then be subject to experimentation in order to optimize and validate the model.

- *“Multi-physics” modeling:*
This approach consists of assembling the components for which behavior is already modeled in the software. This method does not require writing equations and saves a considerable amount of time. Component libraries allow modeling in all physics domains (mechanical, electrical, hydraulic, thermal...) It is therefore very simple to connect the different domains in order to obtain the global model of a system. It is also possible to integrate the 3D model of the modeled structure which allows dynamic behavior and non-linearities related to geometry to be very easily taken into consideration.
- *“Gray box” modeling:*
This approach is a supplement to the “white box” or “multi-physics” model that estimates unknown parameters by comparing the response of the model to the response of the real system. Algorithms are used to automate this task and to assign values to unknown parameters in order to remove the gaps between the performance of the model and that of the real system.
- *“Black box” modeling:*
This approach, also known as identification, consists of associating a mathematical model with a measured experimental behavior. The digital tool enables this task to be automated by suggesting mathematical models for complex identifications. This method provides only a model of the global behavior of the system, without delving into the separate influences of the various performance parameters.

Combining these approaches enables a model to be quickly built and validated through experimentation. This section describes the use of a concrete case of the “Model Based Design” approach. Using the specifications and requirements connected to the design, the various stages of the V-Model will be covered using modern tools for modeling, simulation and validation. The different modeling approaches are compared and the advantages and disadvantages of the various methods are highlighted. The procedure enables the creation of a link between the training process and industrial reality. Significant stages in the “Model Based Design” procedure and the detection of errors in the generation of C code are illustrated (Model in the loop, Software in the loop, Processor in the loop and Hardware in the loop).

II. Industrialization and design cycle of a system

The industrial procedure used by teams of engineers to design a system is a complex process that most often complies with industrial standards such as “V-Model” and associated methods such as “Model Based Design”. Figure 1 shows a simplified representation of the “V-Model,” adapted to present the concepts of this process.

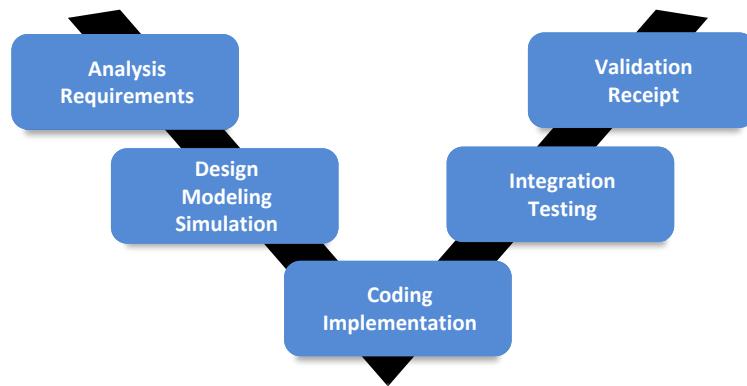


Figure 1: Simplified V-Model

A. Competences of the engineer

In order to use this procedure, the student and engineer must call on 6 broad competences. These competences are the foundation of engineer training and are detailed in Figure 2.

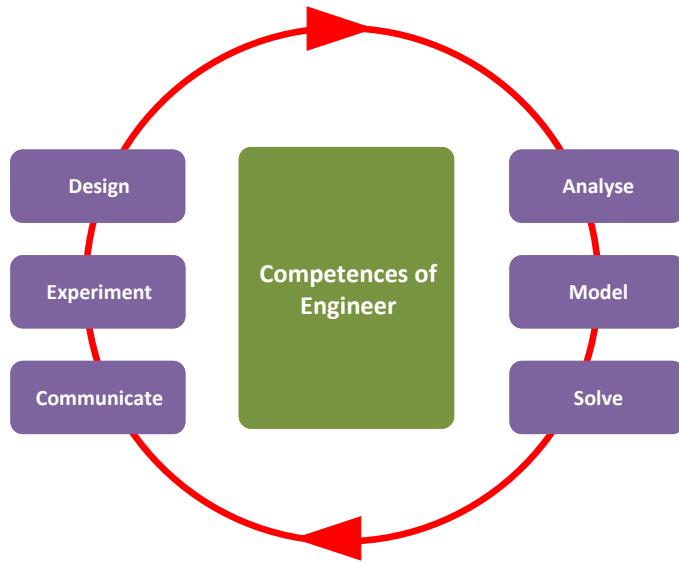


Figure 2: Competences of the engineer

B. The performance triptic^(*)

Figure 3 describes the three types of performance which will be considered and compared in the procedure:

- Performance resulting from specifications and requirements
- Performance resulting from digital model simulation
- Performance resulting from direct measurement of the real system using sensors

Three types of deviations may thus be created and analyzed:

- The gap between specified performance and that measured in the real system
- The gap between performance measured in the real system and performance resulting from digital model simulation
- The gap between performance specified performance and performance resulting from digital model simulation

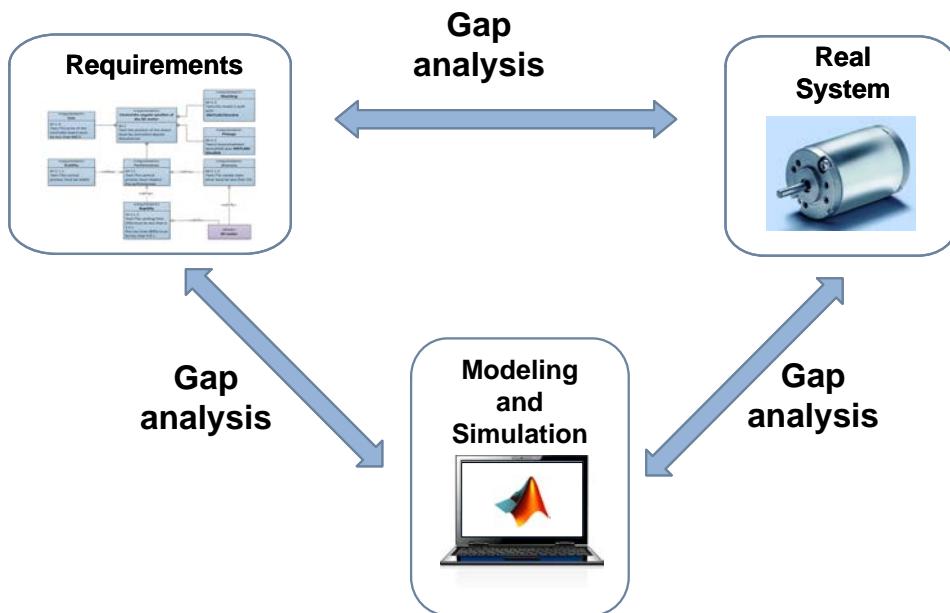


Figure 3: Specifications/real system/ model triptic

(*) You can attend the webinar we have prepared with Nadia Bedjaoui, who presents the use of the engineering procedure using the functionalities of the MATLAB/Simulink software:

<http://fr.mathworks.com/videos/matlab-and-simulink-for-teaching-98093.html>

III. Using the procedure - Introduction to “Model Based Design”

We will apply this procedure to a simple problem to help understand the steps to be taken for the various phases of the V-Model. It is recommended you read this section before starting on the section of this work dedicated to the use of simulation and modeling tools. This will enable the reader to gain perspective on the modeling procedure, to understand the place of the model in the industrial process and to understand the importance of design based on the model, more commonly known as “Model Based Design”.

The chosen example reproduces the design procedure for the operating logic of the servo command in angular position for a DC motor. The objective is to locate the motor axle in an angular position that complies with a position instruction used to meet current performance criteria such as precision, speed or stability.

A. Material architecture of the project

In order to implement the procedure, Figure 4 shows the material environment of the project. It is not necessary to have this material available in order to understand the concepts addressed in this section.

- *A computer with modeling, simulation and piloting software installed:*

The role of the computer is to support the design of digital models, their simulation and the compilation of results. The computer will also be used to create measurement and piloting interfaces in order to control the motor via the motor control card.

- *The motor control card:*

The role of the motor control card is to pilot the motor. It has inputs to receive information from the sensors (angular position of the motor...) and to integrate the instructions coming from the computer, and outputs to supply voltage to the motor and allow it to rotate. Voltage can be modulated in accordance with the programmed command logic. In our example, it will also be a function of the power card.

- *DC motor:*

The DC motor is fed by constant voltage and provides rotation speed. An integrated position sensor sends the angle position of the motor to the command card.

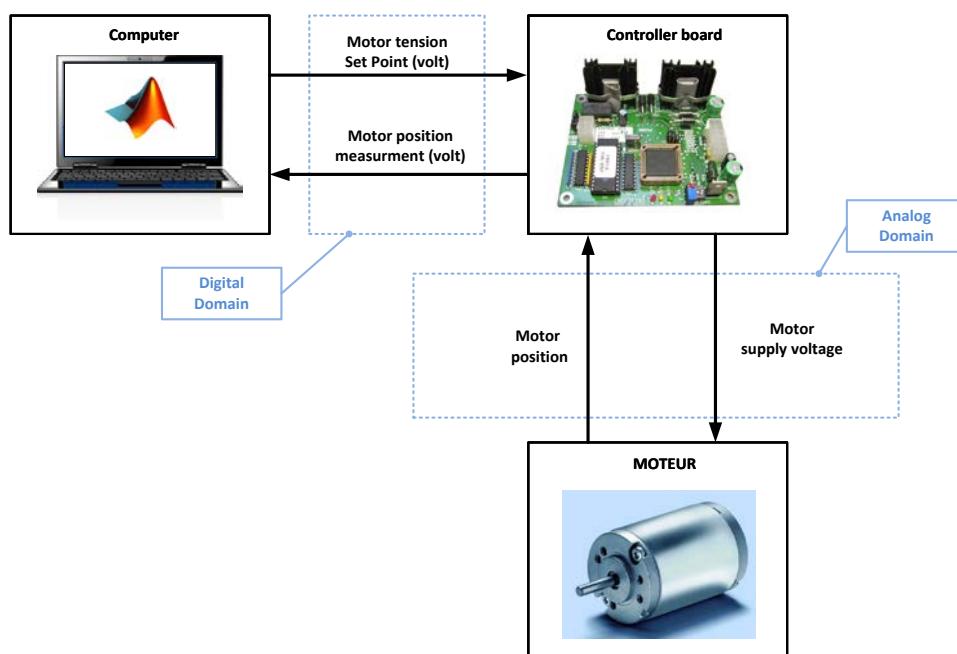


Figure 4: material architecture required to implement the procedure

B. Analysis and Requirements Phase

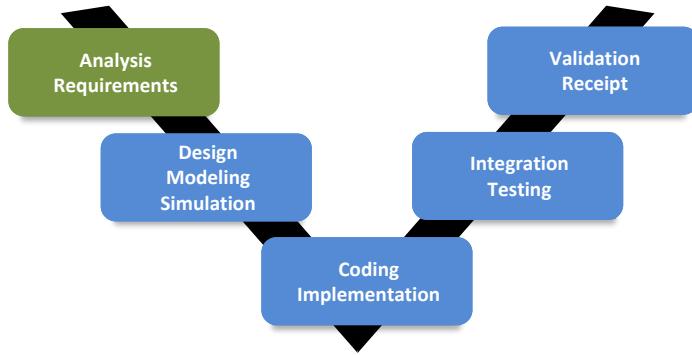


Figure 5: Analysis and Requirements phase

The cycle starts with the **Analysis and Requirements** phase for the system. All requirements that must be included in the system are formalized at this stage. Performance criteria are defined and quantified. All this information is included in the functional specifications. This is analyzed by the design teams who must then provide appropriate technical solutions. The requirements diagram Figure 6, includes all of the requirements for the design of the position servo controller.

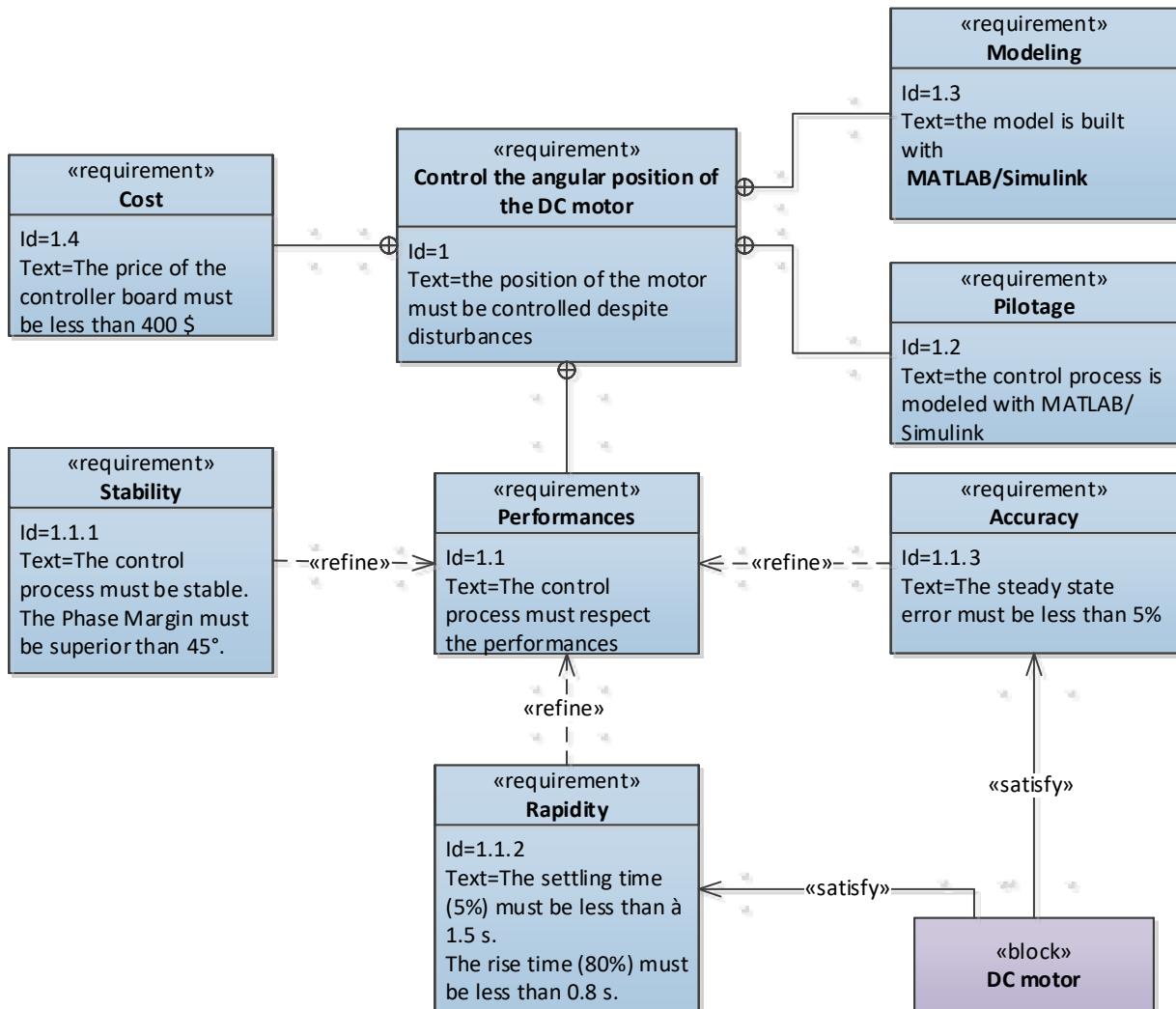


Figure 6: position servo controller requirements diagram

C. Design-Modeling-Simulation Phase

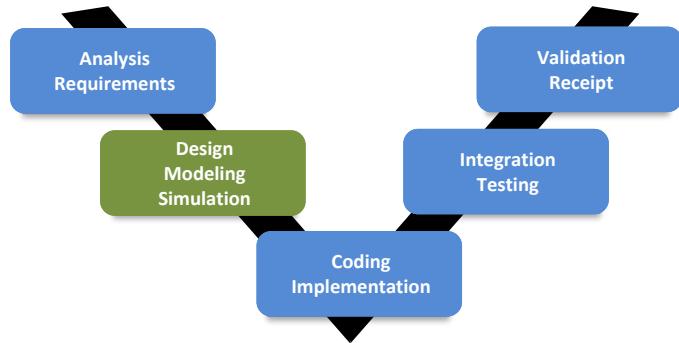


Figure 7: Design, Modeling, Simulation Phase

The **Design-Modeling-Simulation Phase** consists of inventing, innovating and proposing technical solutions to meet the specified requirements. In this phase, the system structure, interconnection of the physical fields (mechanical, electrical, hydraulic, thermal...), control law, etc., are defined. This very complex phase calls for the ability to anticipate behaviors even before the real system is built. This is when the work of modeling begins. Engineers build models to attempt to understand and anticipate the technical choices made. This stage requires high level theoretical and technological knowledge in all of the domains involved. To be able to exploit the models and obtain results, these models must be simulated using the appropriate digital tools.

In industry, it is very expensive to run trials on real equipment. It requires significant human and material resources and severe constraints in terms of deadlines and costs. One of the major objectives is therefore to limit the duration of these tests by optimizing the preparation for this phase. One solution is to replace tests on real materials with trials on digital models that reproduce the behavior of the real system. The "Model Based Design" method is becoming a design standard which will enrich the V-Model process.

This method requires the availability of validated models. Modeling techniques and their mastery by engineering teams are at the heart of the design process. Rapid changes in the functionalities of digital simulation opens new modeling possibilities which accelerate the model creation and validation process. Digital resolution is indispensable to the process and simulation software is a vital part of this process. Its performance and functionalities often determine the speed and effectiveness of the modeling process. Different methods may be used to optimize this phase.

- “White box” modeling:
- “Multi-physics” modeling:
- “Gray box” modeling:
- “Black box” modeling:

These different approaches are not exclusive in the modeling process. On the contrary, they are complementary and are frequently combined in order to achieve the most relevant process. Knowledge and mastery of these concepts by students and engineers enables them to achieve competence in the modeling process. In all cases, experimentation and measurement facilitate the validation of the model behavior.

In our example, we need to create the digital model of a DC motor to carry out the entire command control phase on a model after it is validated.

Modeling of a DC motor requires establishing the relationship between the voltage at the terminals coupled to the motor and its rotation speed.

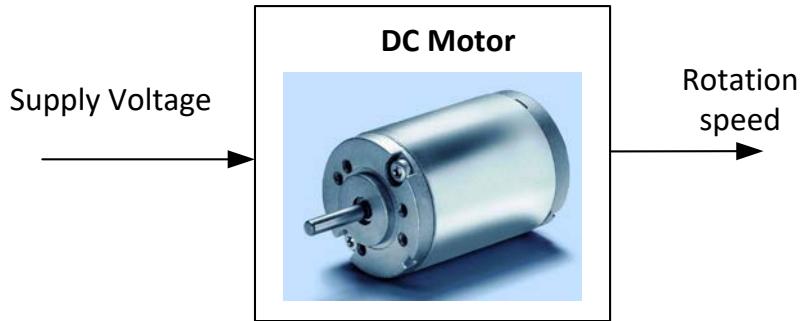


Figure 8: input/output relationship for a DC motor

1. "White box" modeling

The electrical, mechanical and coupling equations of a motor represent the knowledge model for the DC motor and are detailed in Figure 9:

$$\begin{cases} u(t) = e(t) + R i(t) + L \frac{d i(t)}{dt} \\ e(t) = K_e \omega_m(t) \\ J \frac{d \omega_m(t)}{dt} = C_m(t) - C_r(t) - f \omega_m(t) \\ C_m(t) = K_m i(t) \end{cases}$$

$u(t)$: motor supply voltage (V)
 $e(t)$: electromotive force (V)
 $i(t)$: armature current (A)
 $C_m(t)$: motor torque (N.m)
 $C_r(t)$: resistant torque (N.m)
 $\omega_m(t)$: angular speed (rad/s)
 R : armature resistor of the motor (Ω)
 L : armature inductor of the motor (H)
 J : inertia of the motor ($\text{kg} \cdot \text{m}^2$)
 f : viscous friction parameter (N.m.s)
 K_t : torque constant (N.m/A)
 K_e : electromotive force constant (V.s/rad)

Figure 9: equations for the behavior of DC motors

Among the parameters that characterize the behavior of the motor, some are well known and others remain undetermined. The inductance L and the resistance R at the coupling are given in the technical specifications for the motor, as are coefficients K_e and K_t . On the other hand, viscous friction f is not necessarily known, in particular if the motor is later coupled to a mechanism. Therefore, parameter f must either be ignored or an estimate must be made to attempt to obtain valid results with the model.

After applying the Laplace transform to these equations, we can easily obtain a digital model in the form of a block diagram which we can model and resolve using the digital simulation software (Figure 10). This resolution principle is based on the principle of causality. The dynamic behavior of a system is characterized by a block containing, for example, the transfer function of the system, with an input and an output. Information circulating in the connections between two blocks is a directed digital signal. The block output is calculated digitally by determining, for each calculation step, the transformation of the input signal controlled by the content of the block. This approach, which is widely used in industry, requires complete knowledge of the physical laws that characterize the behavior of systems.

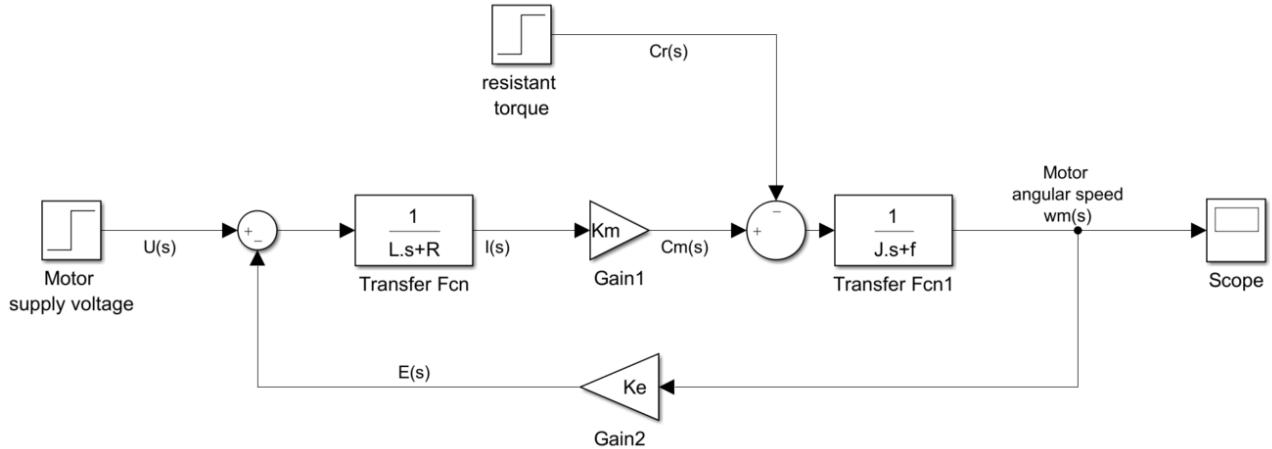


Figure 10: modeling the DC motor in the form of a block diagram using Simulink

It is now possible to replace the system by its model as represented in Figure 11.

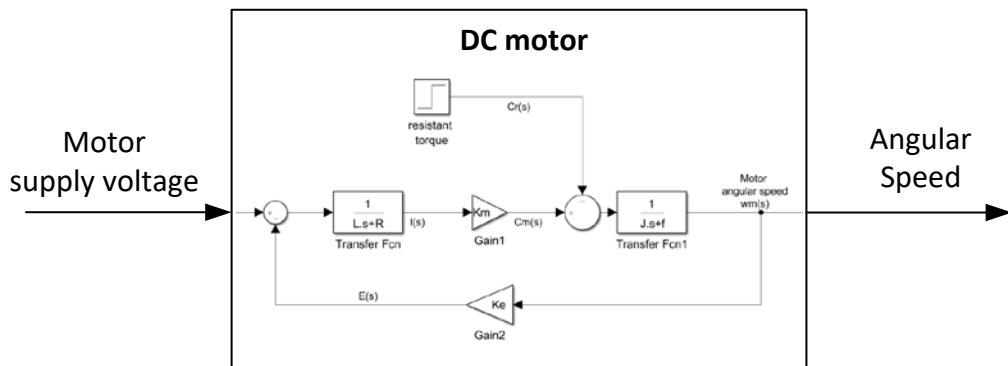


Figure 11 : "White box" modeling of the DC motor

2. "Multi-physics" modeling

"Multi-physics" modeling offers an acausal approach to modeling and enables it to produce a model by assembling components from predefined libraries. The physical behavior of components is directly taken into consideration by the software, and it is thus possible to model a system without having to write the differential equations that characterize its behavior.

Connections between two components are not directed, have a physical meaning and transmit a higher degree of information than connections in causal modeling. Connections may be an electric wire (transfer of current and voltage information), a drive shaft (transfer of torque and angular speed information), the end of the cylinder rod (transfer of force and linear speed information) ... The calculation principle is supported by a power balance at each node of the model and is not based on the principle of causality, giving it the name acausal modeling.

Multi-physics modeling of a DC motor is represented in Figure 12.

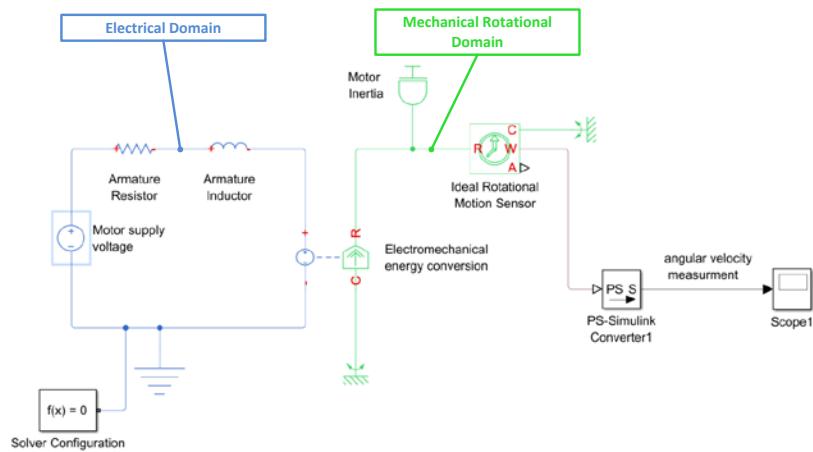


Figure 12: model created through assembling the components of the DC motor

It is now possible to replace the system by its model as represented by Figure 13.

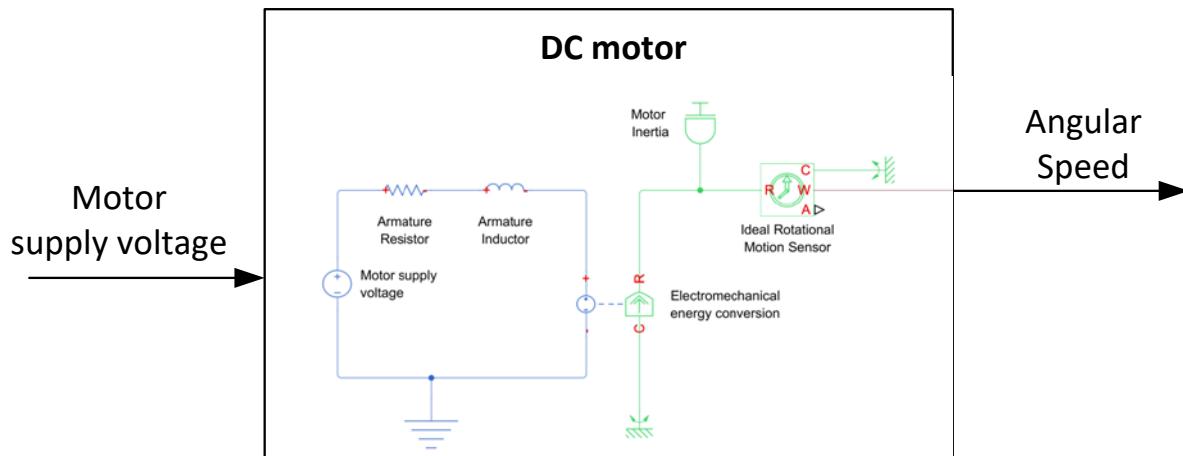


Figure 13: acausal multi-physics modeling of the DC motor

3. Model simulation

The simulation and resolution of models by simulation software requires knowledge of the values of all digital parameters. In our example, all of the parameters are known except for the f parameter that characterizes frictions. It will therefore be considered null in the first approximation which ignores the influence of frictions.

Regardless of the approach chosen, “white box” or “multi-physics,” the results obtained by digital simulation are identical (Figure 14). In both cases, the physical phenomena used are the same, the difference is in the method used to build the model and the digital resolution method.

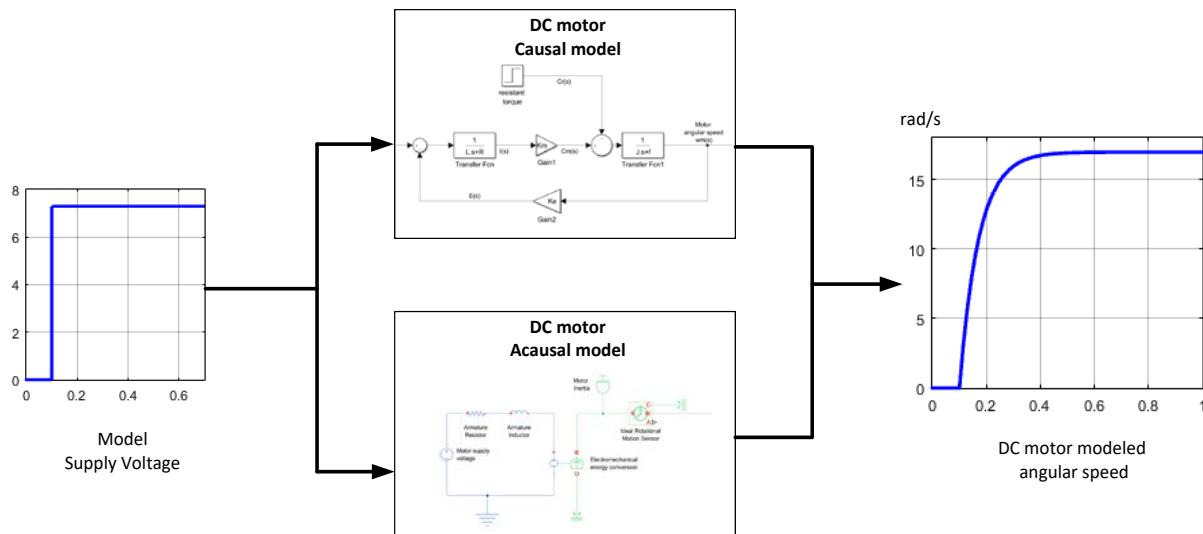


Figure 14: resolution and visualization of results

The results obtained provide a first approximation of the change in the motor rotation speed as a function of time. In general, this approach is very often insufficient and the only means of verifying the validity of the model is to evaluate the differences between modeled performance and performance measured on the real system. By subjecting the terminals coupled to the real motor to an identical voltage, it is possible to discover the real rotation speed of the motor, as shown in Figure 15.

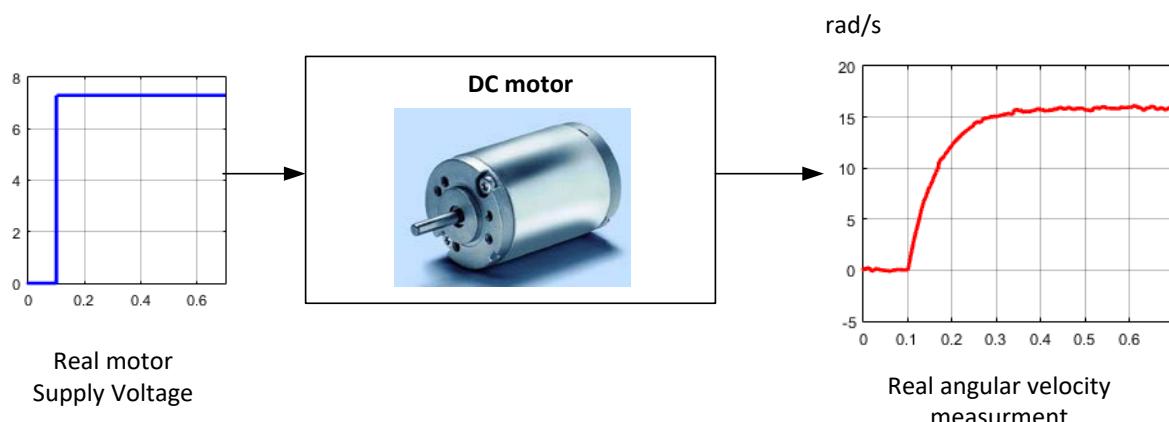


Figure 15: measurement of the evolution of the rotation speed of the real motor as a function of time

4. Comparison of simulated and measured performances

Simulated and measured performances are thus compared in Figure 16.

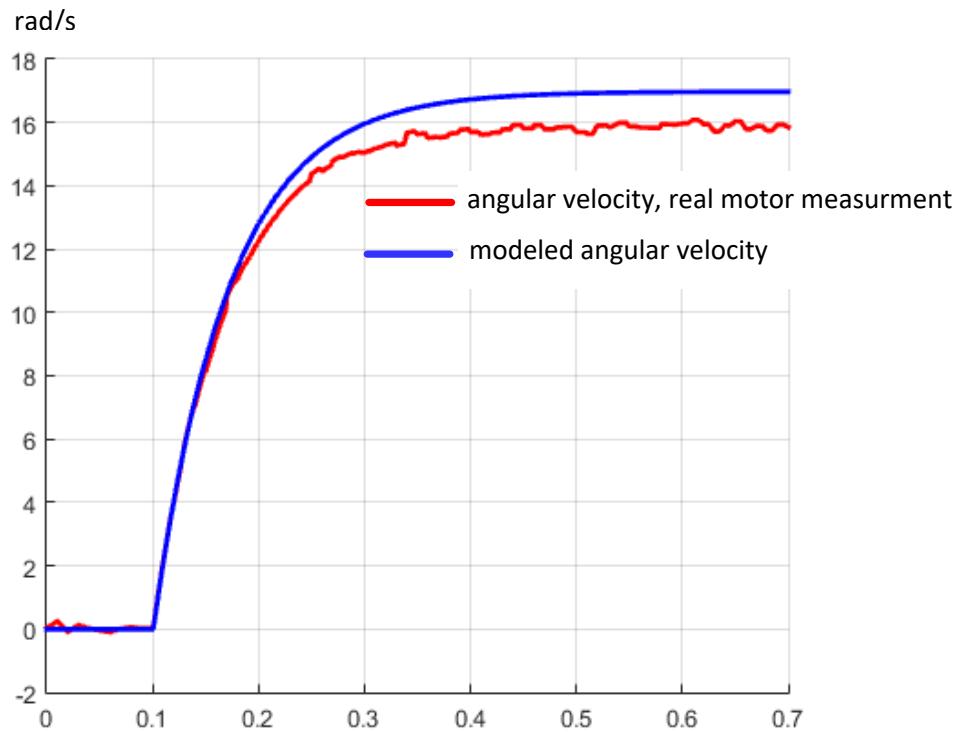


Figure 16: evaluation of differences between simulated and measured performances

Generally at this stage, behavior differences between the model and the real often become very obvious. Therefore, these gaps must be evaluated and paths for improving the modeling must be found by considering any new phenomenon or by adapting the value of some parameters.

In our example, the analysis of the differences shows that the model is not validated. Failure to consider frictions is the source of this. Unknown parameters must be therefore be evaluated, the f friction parameter in this instance. The “gray box” approach will enable us to complete the modeling process by offering a method for determining unknown parameters.

5. "Gray box" modeling

The principle for this modeling is to estimate the unknown parameters of the model by comparing simulated performance and measured performance. An algorithm integrated into the simulation software will do the work of analyzing these gaps and suggest possible values for unknown parameters that will cancel out these gaps. The method involves conducting a trial on the real system by increasing the signal applied at input to the system and the signal obtained at output (Figure 17).

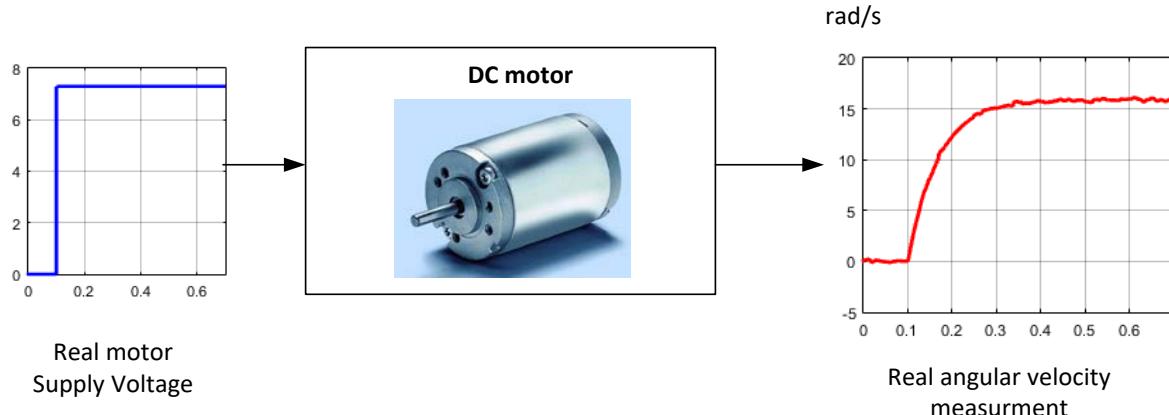


Figure 17: trial conducted to estimate unknown parameters

Then, an identical input signal is introduced at the model input and the software will provide possible values for the unknown parameters to remove the gaps between the real machine and the model. The model will seek to attain the target response, which is that of the real system (Figure 18). It is also possible to use the causal model instead of the acausal model to achieve this optimization.

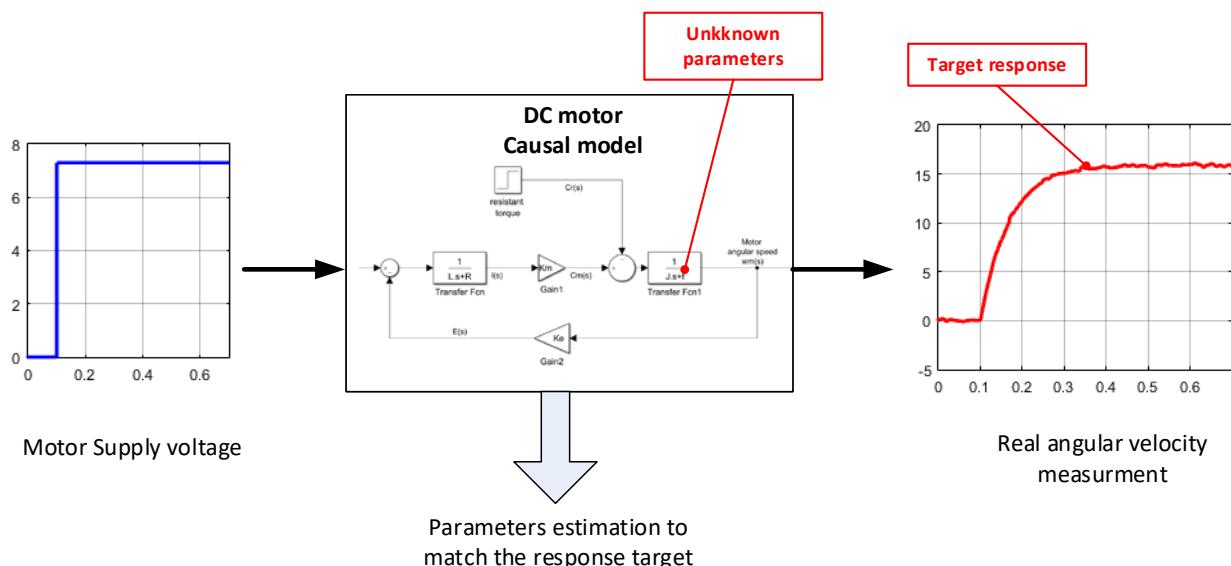


Figure 18: illustration of the principal of parameter estimation

Figure 19 shows the target response in red that will seek to attain the parameters estimation algorithm. The response of the model before estimation of the parameters is represented by the blue curve and the response of the model after estimation of the parameters is shown by the green curve. This is clearly of interest and facilitates the obtaining of a validated model for which the behavior is very close to that of the real system. Gaps between measured performance and simulated performance have been eliminated.

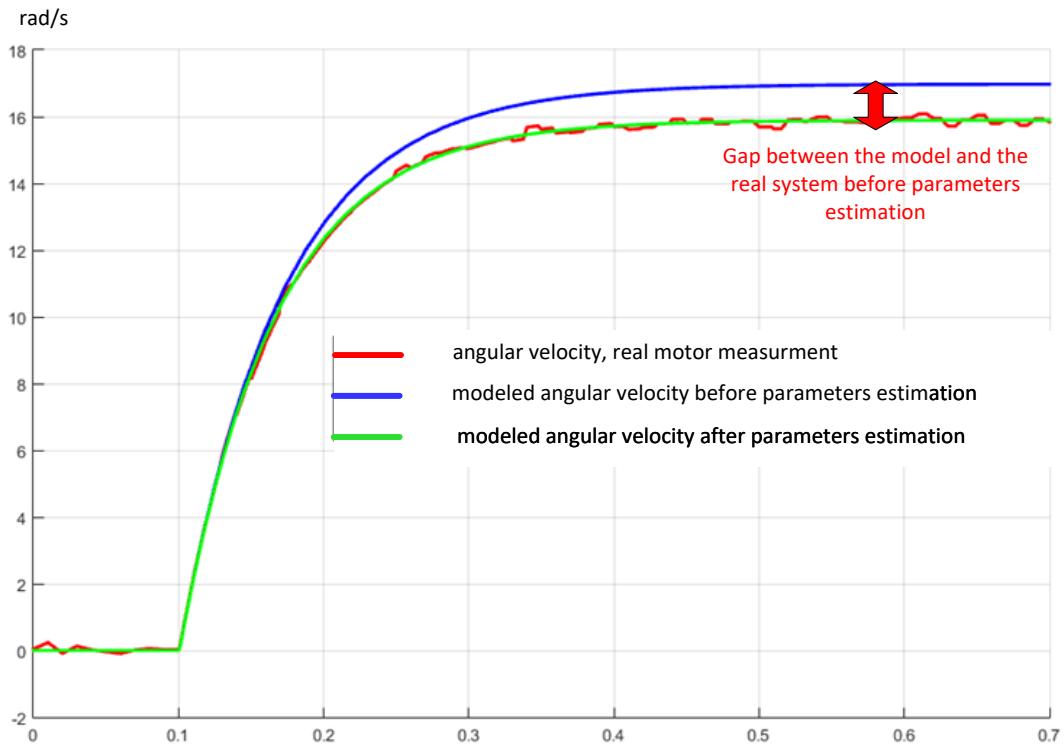


Figure 19: response of the model before and after estimation of unknown parameters

At the end of this phase, the project has validated models. It is then possible to replace the equipment (the motor in this instance) with their model. Implementation of the design phase and control adjustment will be done only by using the simulation software and working on the model. Tests on real equipment will be done after the model performance reaches the desired performance in the specification.

We can now use the validated model of the motor to build the servo controller for the DC motor position represented in Figure 20.

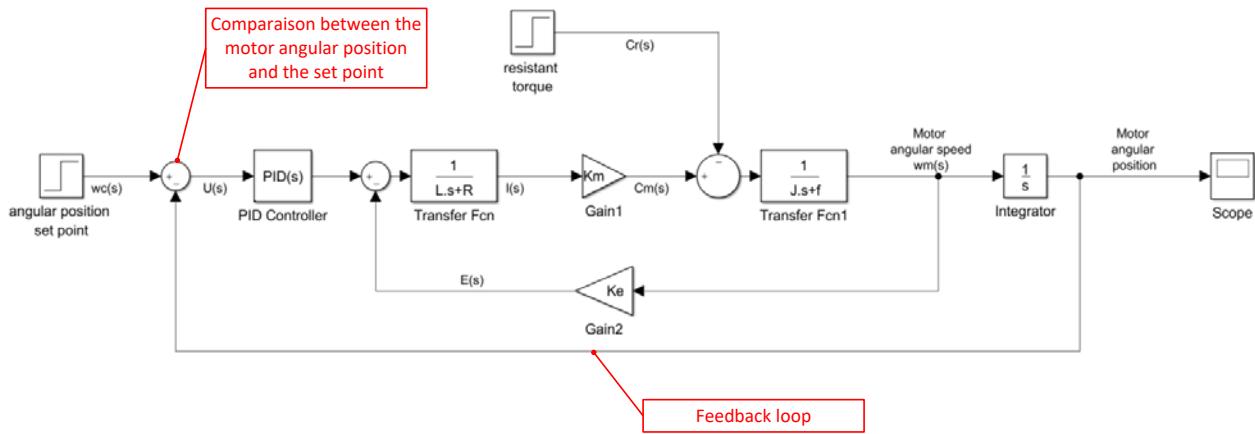


Figure 20: modeling the servo controller for the motor position

First, the compensator is not activated and the position servo controller response is shown in Figure 21.

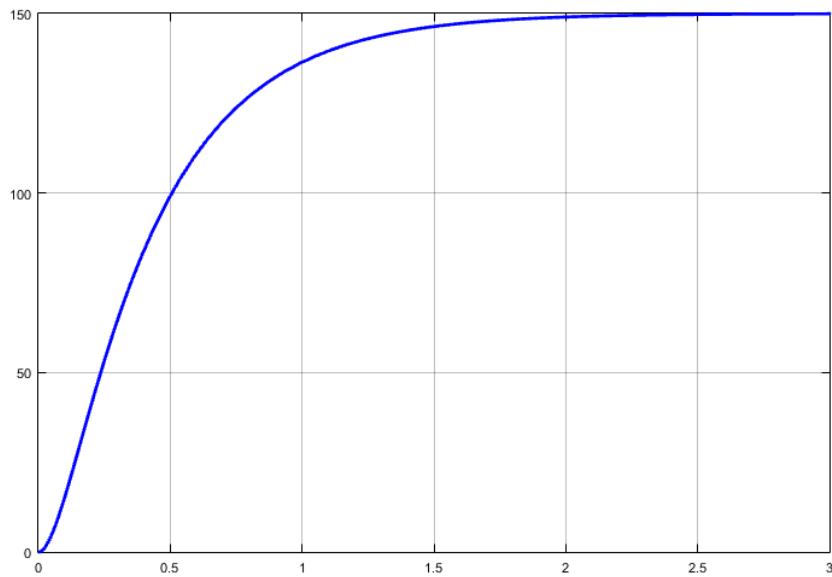


Figure 21: DC motor position response without correction

The requirements diagram calls for the following performance criteria:

- Settling time within 5%, less than 0.8 s
- Rise time of 80%, less than 0.4 s
- Steady state error less than 5% of required set point

The modeled servo controller performance does not appear to comply with specified performance. The settling time within 5% is of the order of 1.3 s and the rise time of 80% is of the order of 0.7 s. Therefore, the motor control must be optimized by adjusting the PID controller. The simulation software then suggests numerous functionalities that will facilitate implementing the procedure. Figure 22 illustrates the use of the method of adjusting a compensator in the frequency field using specific tools.

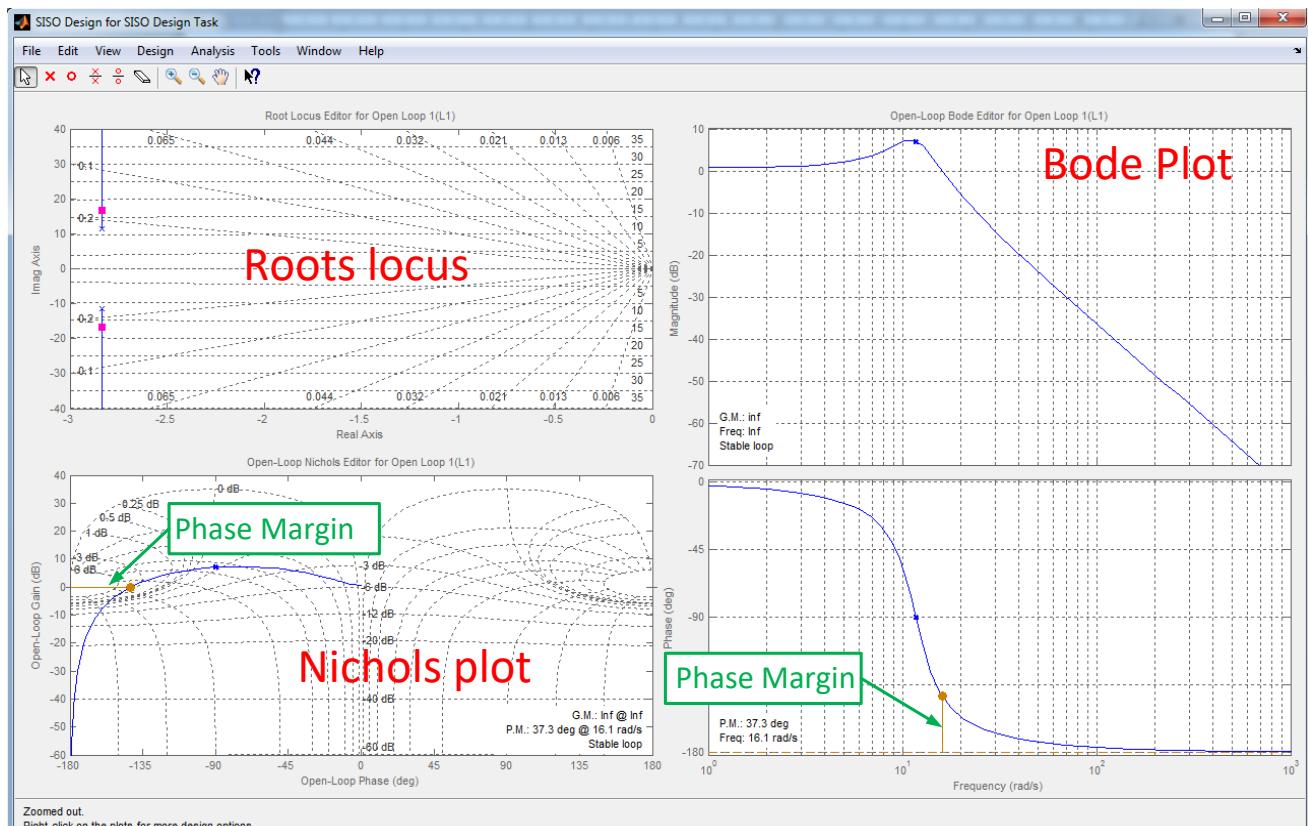


Figure 22: illustration of the use of a command control tool for adjusting a compensator

These methods will be largely developed in the follow-up to this work and call for the completion of a number of trials on the model. As the servo controller model was built on the basis of the validated motor model, the results obtained in simulation will be very close to the results measured on the real system.

At the end of this adjustment phase, the compensator is set to obtain the response shown in Figure 23.

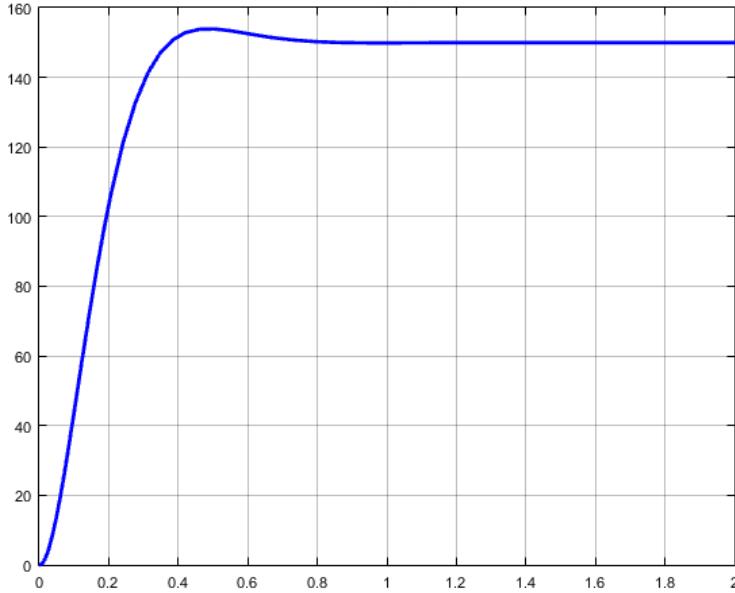


Figure 23: DC motor position response with PID correction

Settling time within 5% is now of the order of 0.3 s and the rise time at 80% of the order of 0.25s. The modeled performance is in compliance with the specified performance and the **Coding Implementation** phase can begin.

6. Model-in-the-loop (MIL)

This stage is located entirely within the domain of simulation and corresponds to the simulation of the complete model previously described. The model is described entirely in MATLAB code entirely on the computer with which the model was built.

Model-in-the-loop (MIL)

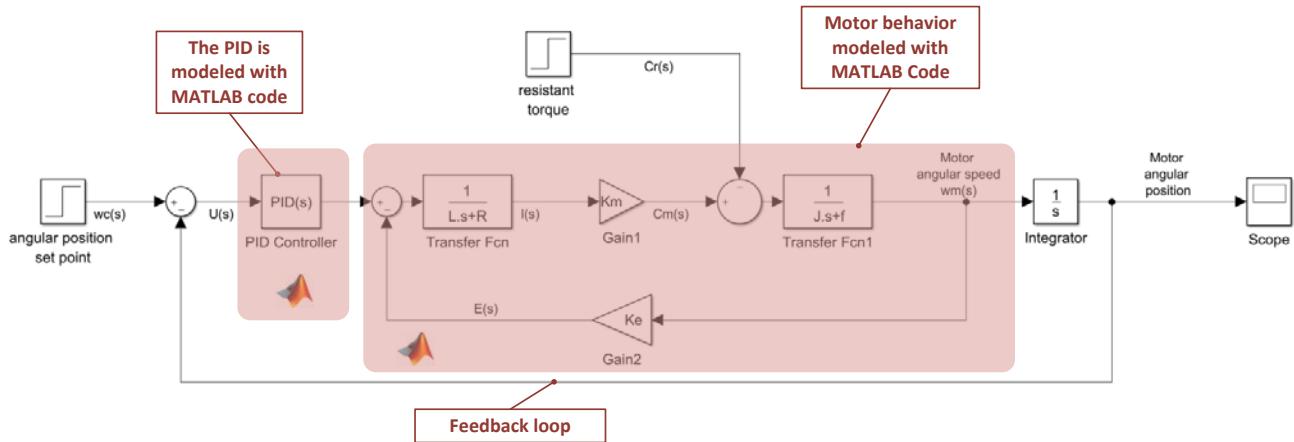


Figure 24: Model-in-the-loop (MIL)

D. Coding Implementation Phase

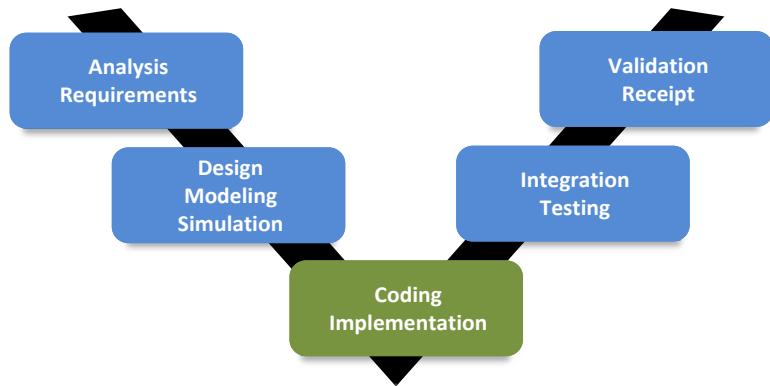


Figure 25: Coding Implementation Phase

This phase enables progressive migration from the simulation domain to the test phase on real equipment.

At this stage, it is important to return to the structure of the validated model which will be finalized at the end of the **Design Modeling Simulation** phase. Figure 26 shows the two parts of the model that must be clearly distinguished in order to understand the stages that will follow.

- the part of the model corresponding to the compensator, which was the basis of the project design objectives.
- the part of the model corresponding to the motor that provides a validated model of the actuators.

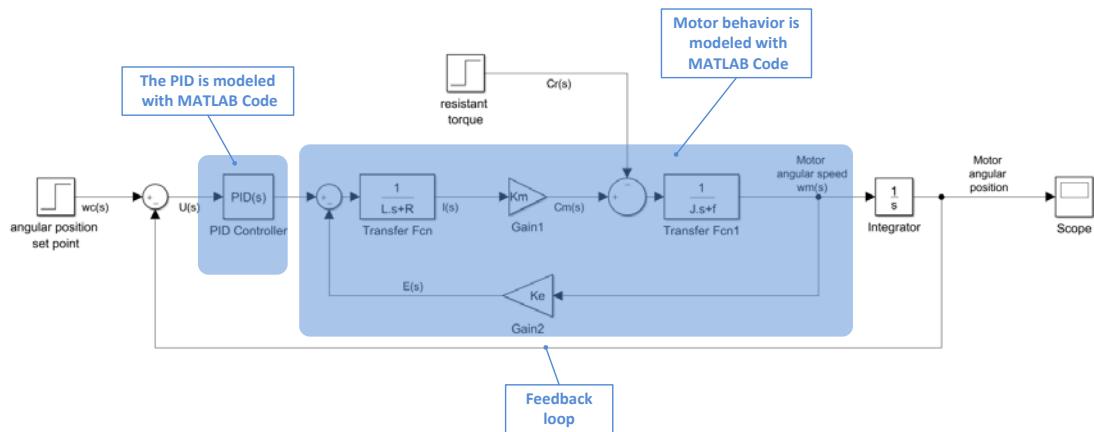


Figure 26: structure of the model at the end of the Design Modeling Simulation phase

This phase enables the translation of the parts of the model into C programming language. C is the programming language used by the control card as it cannot directly interpret MATLAB code. One method is to isolate the part of the model we want to replace with C Code and create a list of the inputs and outputs for this part of the model. It is possible to complete this coding manually. In this case, a team of coders specialized in C language will execute the programs that will generate the same relationship between inputs and outputs as that obtained with the MATLAB code. This approach is long and requires high level coding and programming competences.

Simulation software that also enables the generation of C Code automatically and MATLAB offers interfaces enabling automation of this process in order to save precious time.

Whether they are automated or manual, the coding process in C language can generate errors and the behavior of the model coded in C language may differ from its behavior when coded in MATLAB. In order to detect and correct these errors before they can impact the test phase on real equipment, it is necessary to implement two configurations of functioning:

- Software-in-the-loop (SIL)
- Processor-in-the-loop (PIL)

1. Software-in-the-Loop (SIL)

In this configuration, part of the model is replaced by C Code which will be executed on the computer. This will enable verification that the generation of C Code does not cause any errors in terms of translation of C Code into MATLAB code.

In our example, we are going to replace the compensator coded in MATLAB by a C function which will generate C Code to replace the MATLAB code and execute this C Code on the computer (Figure 27).

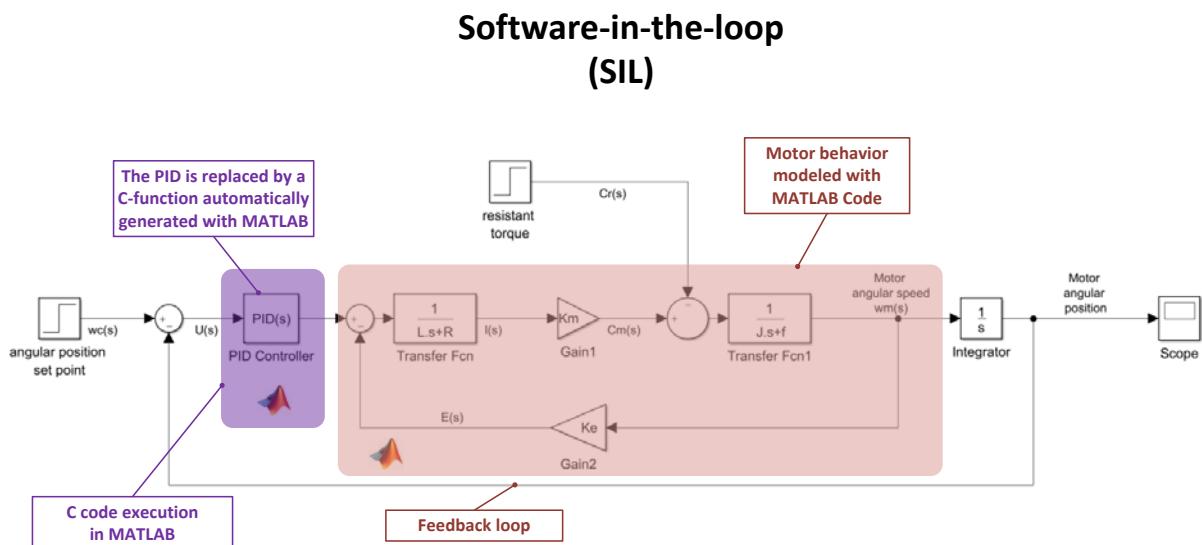


Figure 27: Software-in-the-loop (SIL)

It is also necessary to verify that the behavior of the model is the same, whether it is piloted with the compensator coded in MATLAB code or with the compensator coded in C Code. Once this verification is done, the designers can exclude any error due to the generation of C Code.

2. Processor-in-the-loop (PIL)

This configuration enables errors to be avoided in the execution of C programming language. In effect, the conditions for the execution of C language depend largely on the capacities and performances of the target materials on which they are executed. In software-in-the-loop mode, the code is executed on the computer's processor, in processor-in-the-loop mode, the code is executed on the target material which loads the code under real functioning conditions. These two processors may generate the execution of the same program in different ways. The modeled compensator is then replaced by C language which will be executed directly on the command card. In this phase, the processor of the card pilots the model of the motor.

Processor-in-the-loop (PIL)

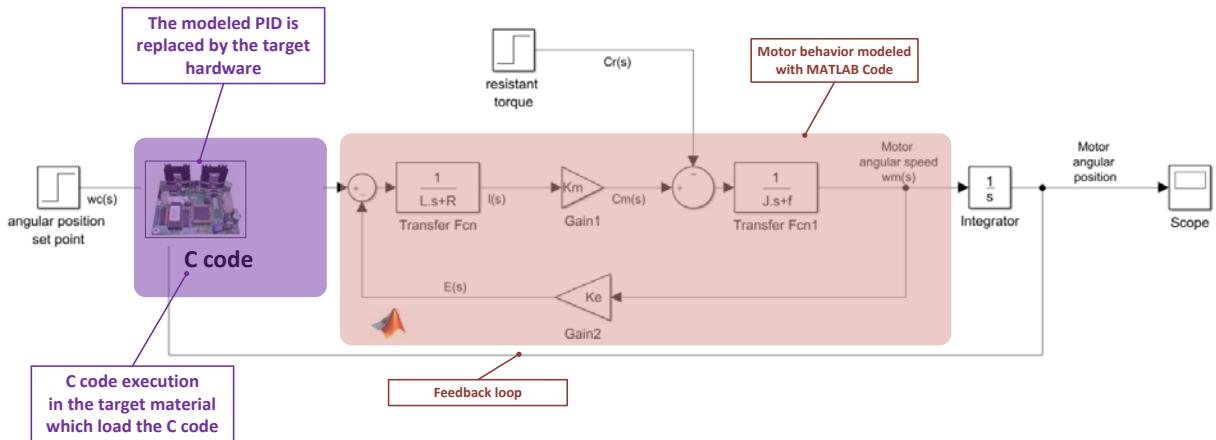


Figure 28: Processor-in-the-loop

At the end of these two phases, errors in generation and execution of C language have been corrected and the test phase on the real actuators can begin.

E. Integration and Testing Phase

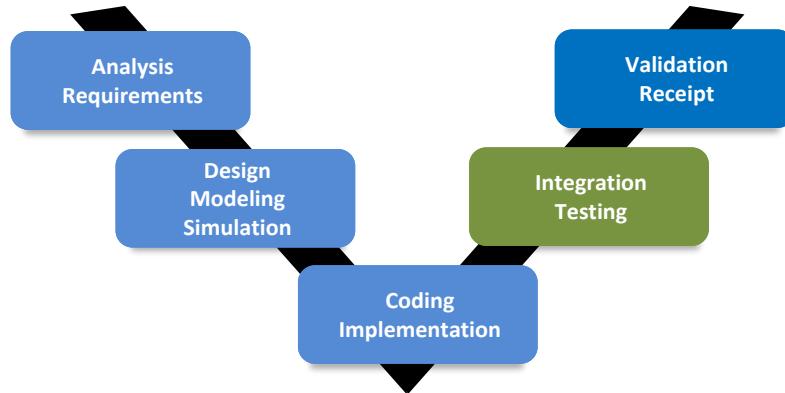


Figure 29: Integration and Testing Phase

1. Hardware-In-the-Loop (HIL)

This stage replaces the motor model with the real motor. The compensator and the associated calculations can be executed either by the computer (external mode), or by the command card which functions autonomously (embedded mode).

HIL - external mode (Figure 30 and Figure 31)

In this phase, all of the calculations are executed on the computer and the command card functions only in power interface. The calculation functions of the command card processor are not called on.

Hardware-in-the-loop

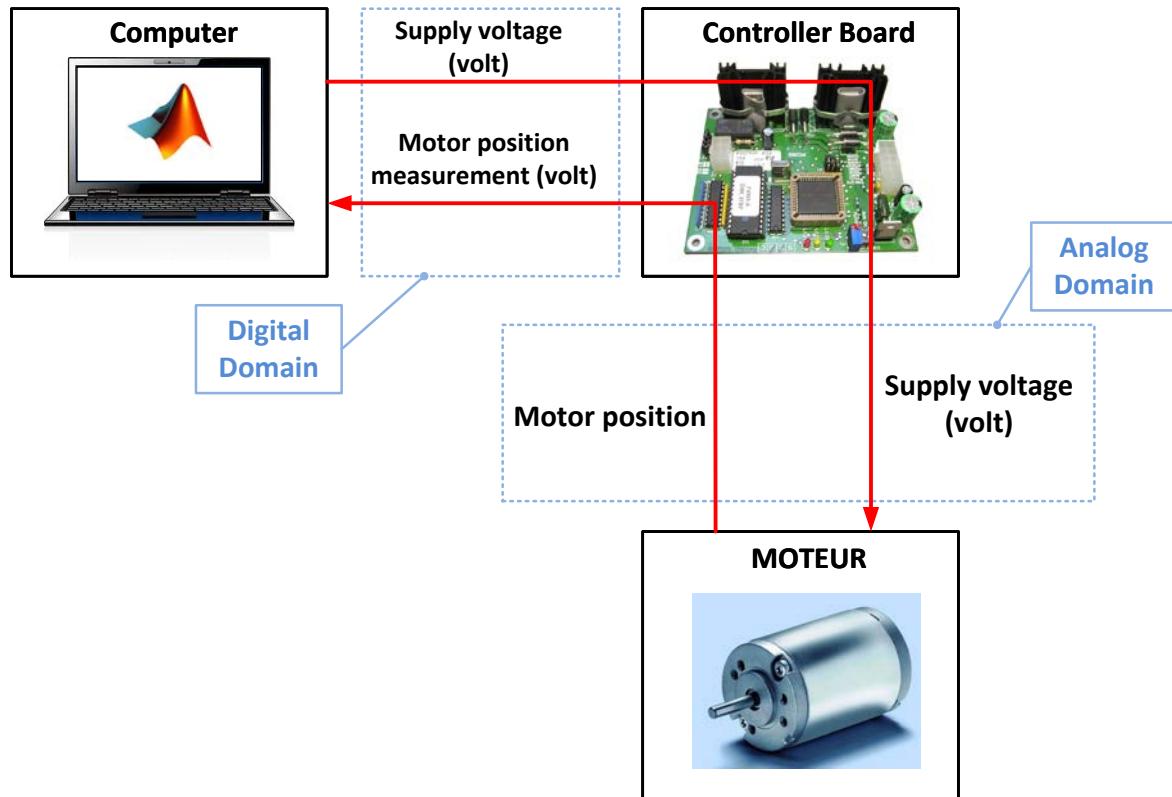


Figure 30: materials architecture for hardware-in-the-loop in external mode

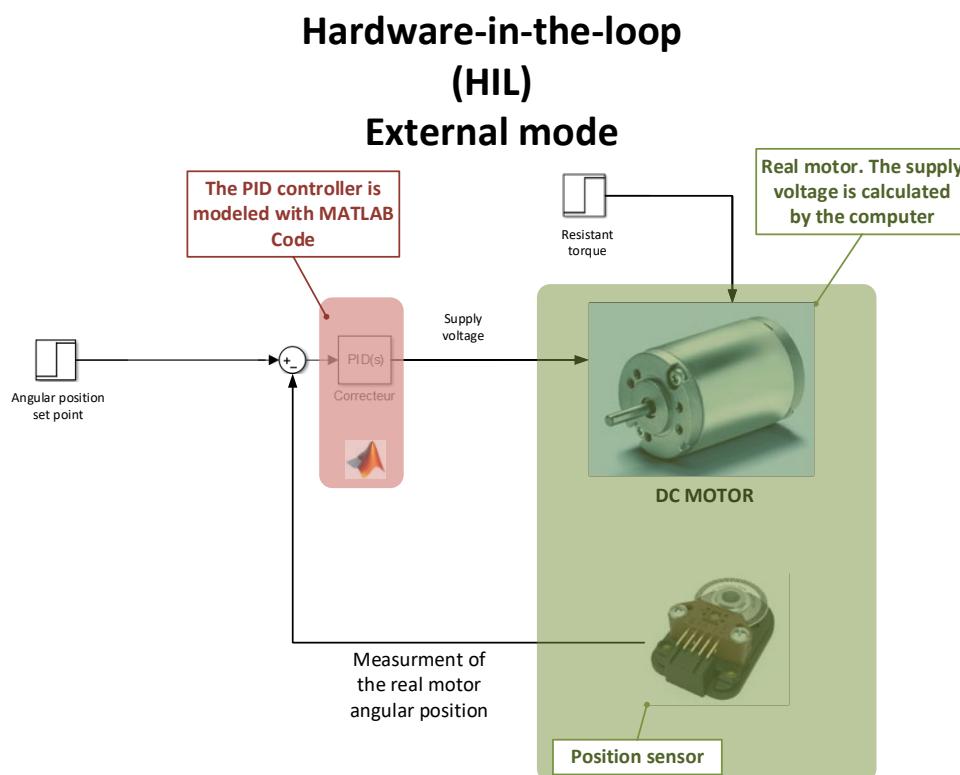


Figure 31: hardware-in-the-loop in external mode: integration of materials into the loop

HIL - embedded mode (Figure 32 and Figure 33)

In this phase, all calculations are executed on the command card. Command logic is executed on the processor of the command card.

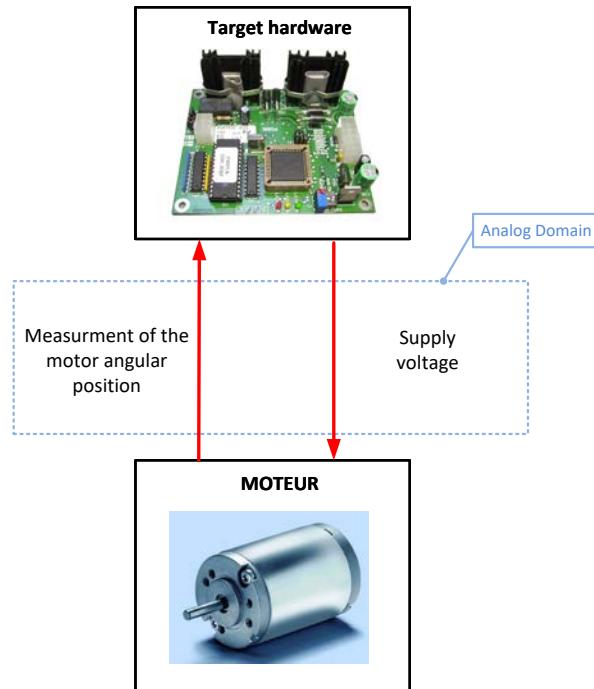


Figure 32: materials architecture for Hardware-In-the-Loop in embedded mode

Hardware-in-the-loop (HIL) Embedded mode

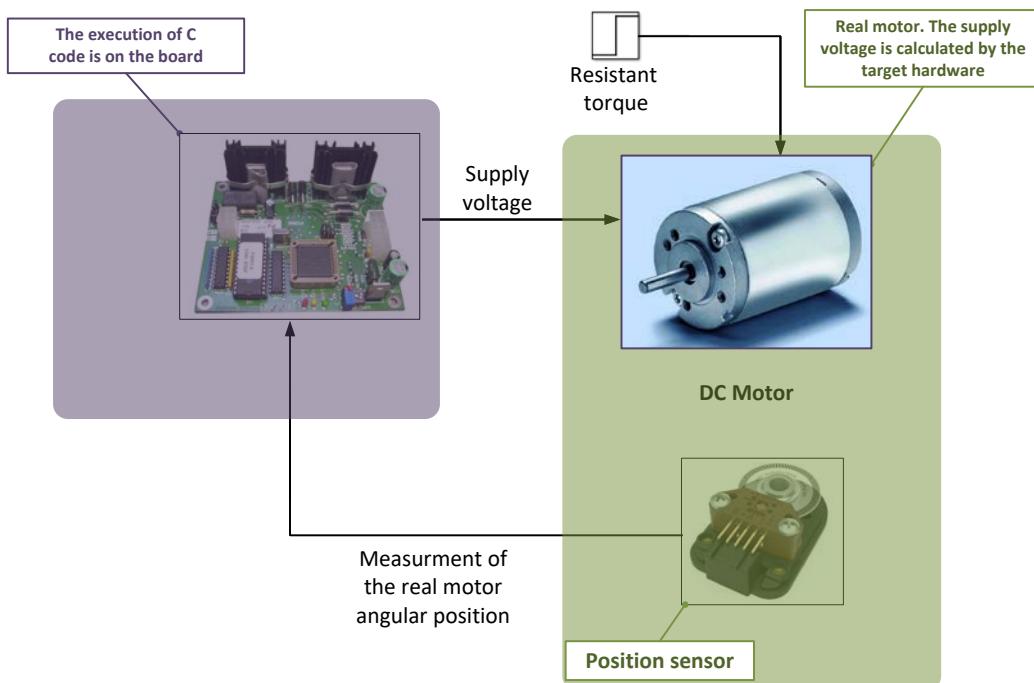


Figure 33: Hardware-In-the-Loop embedded mode: autonomous functioning of materials

One or the other of these configurations enables verification that the perfected compensator will allow the performance specified for the real equipment to be attained. It is then possible to increase the effective speed of the motor on Figure 34. It is then possible to go on to the Validation phase.

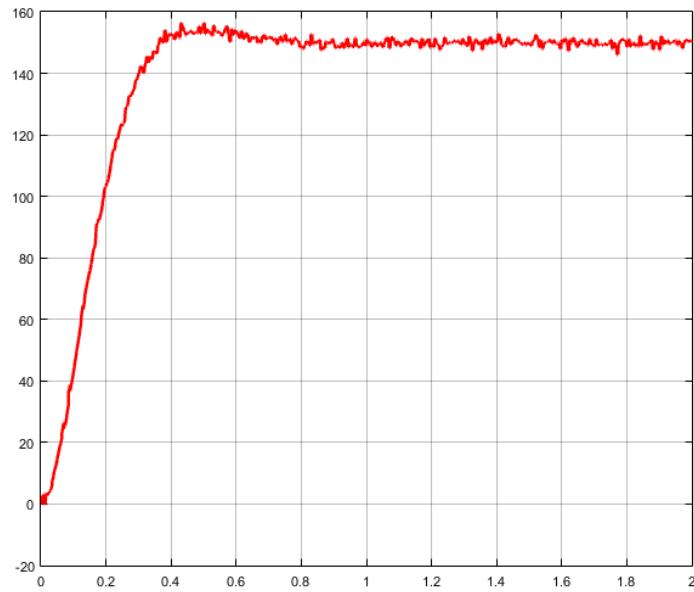


Figure 34: View of the speed of the motor output at the end of the Hardware-in-the-loop phase

F. Validation Receipt Phase

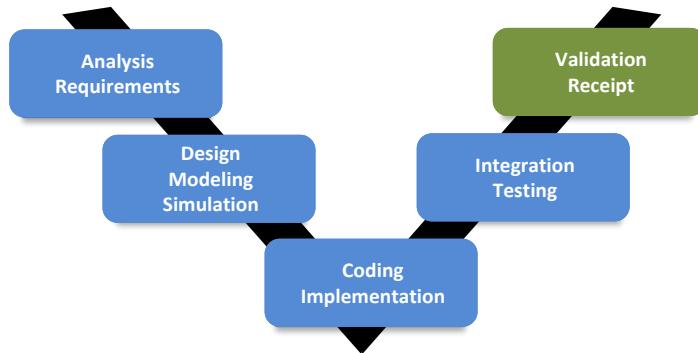


Figure 35: Validation Receipt Phase

In this phase, the correspondence of the measured performance with specified performance still remains to be verified. The analysis in Figure 34 allows us to see that the response time at 5% is in the order of 0.3 s and the rise time at 80% is in the order of 0.25 s. Real performance is compliant with modeled performance and specified performance.

The test phase has been reduced to the bare minimum and the specifications have been met.

Chapter 2: Introduction and presentation of modeling tools

I. MATLAB-Simulink software

MATLAB-Simulink software offers numerous tools enabling the use of global multi-physics modeling. The first stage is to select a palette of appropriate tools that enable the modeling of systems as part of engineer training.

It is preferable, before beginning to work with these tools, to have a global vision of their functionalities and of the causal or acausal approach. The relevance of the modeling procedure lies, above all, in the choice of the appropriate tool for modeling the various parts of the system. Structural mastery of these tools and of their potential will ensure the success of the modeling process.

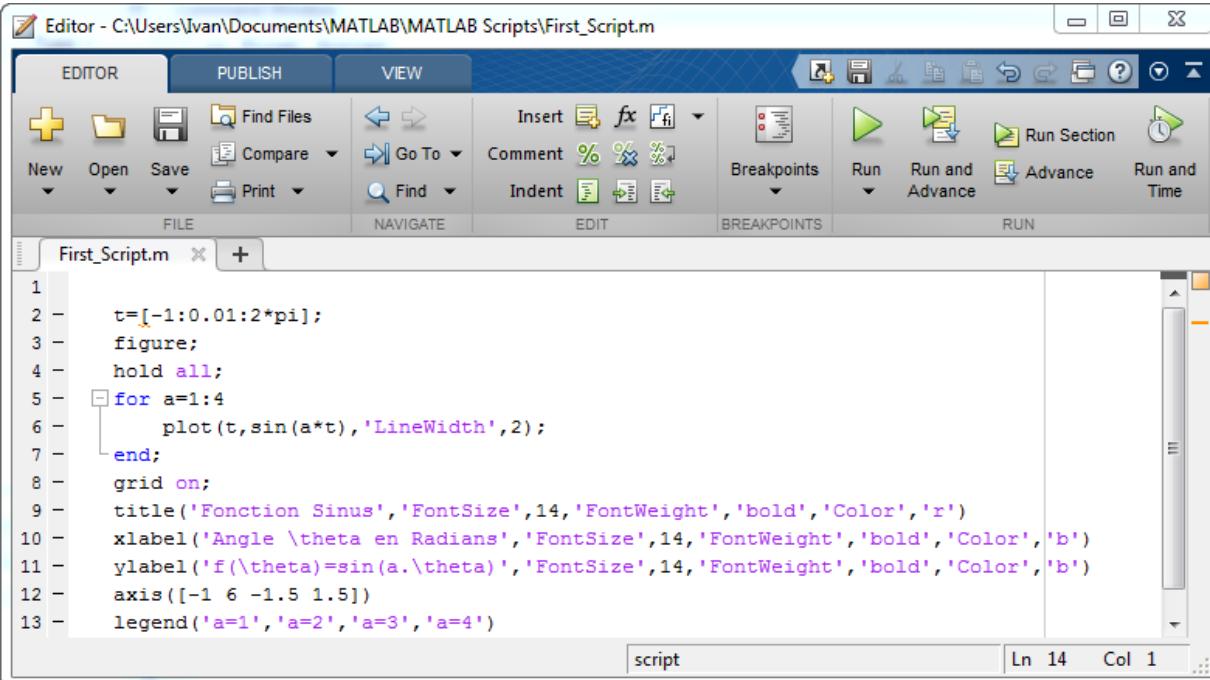
First, we will describe the functionality of these tools. Then, we will make the connection between these tools and the descriptor "Energy chain/information chain," in order to implement a global modeling strategy for systems.

A. Description and hierarchy of tools used.

1. MATLAB

MATLAB is a powerful calculation tool that enables the instructions to be input in the form of command lines. **MATLAB** has libraries of functions grouped into "Toolboxes" relating to a knowledge domain. **Control System Toolbox** for the command control of systems, **Signal Process Toolbox** for the processing of a signal...

Figure 36 gives an example of a program written in MATLAB (script) and Figure 37 shows the obtained results.



The screenshot shows the MATLAB Editor window with the title bar "Editor - C:\Users\Ivan\Documents\MATLAB\MATLAB Scripts\First_Script.m". The menu bar includes "EDITOR", "PUBLISH", and "VIEW". The toolbar contains icons for New, Open, Save, Find Files, Compare, Go To, Find, Insert, Comment, Indent, Breakpoints, Run, Run and Advance, Run Section, and Run and Time. The main workspace displays the following MATLAB script:

```
1 t=[-1:0.01:2*pi];
2 figure;
3 hold all;
4 for a=1:4
5 plot(t,sin(a*t),'LineWidth',2);
6 end;
7 grid on;
8 title('Fonction Sinus','FontSize',14,'FontWeight','bold','Color','r')
9 xlabel('Angle \theta en Radians','FontSize',14,'FontWeight','bold','Color','b')
10 ylabel('f(\theta)=sin(a.\theta)','FontSize',14,'FontWeight','bold','Color','b')
11 axis([-1 6 -1.5 1.5])
12 legend('a=1','a=2','a=3','a=4')
```

The status bar at the bottom right shows "script" and "Ln 14 Col 1".

Figure 36 : example of script written in MATLAB language

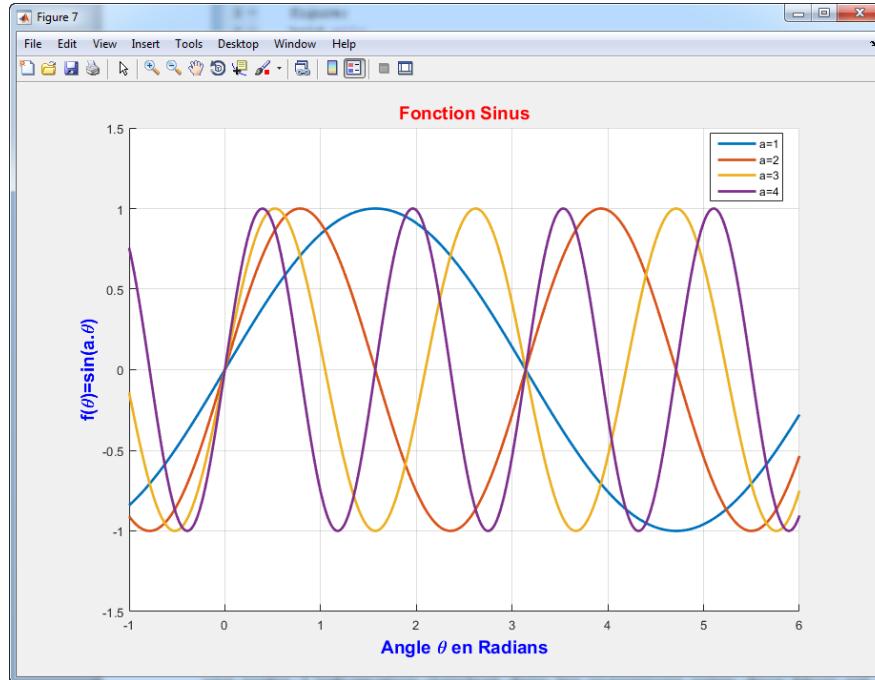


Figure 37: the result obtained after execution of the script

2. Simulink

Simulink is the **MATLAB** graphic interface that enables it to generate code and the indispensable syntax for the uploading of the command lines. **Simulink** possesses libraries of blocks, grouped into Blocksets (Simulink Control Design for the command control of systems, for example). **Simulink** offers a causal approach to modeling. The dynamic behavior of a system is characterized by a block containing, for example, the transfer function of the system, with an input and an output. Information circulating in the connections between two blocks is a directed digital signal. The block output is calculated digitally by determining, for each calculation step, the transformation of the input signal controlled by the content of the block. Simulink also offers a complete palette of highly evolved tools for the command control of servo controller systems.

This widely used approach requires thorough knowledge of the physical laws of the physics phenomenon that characterize the behavior of the systems. Any modelling phase begins with the writing of differential equations characteristic of the physical phenomenon studied, and with obtaining transfer functions for all of the subsystems of the system studied.

Figure 38 shows a **Simulink model** and Figure 39 shows the results obtained:

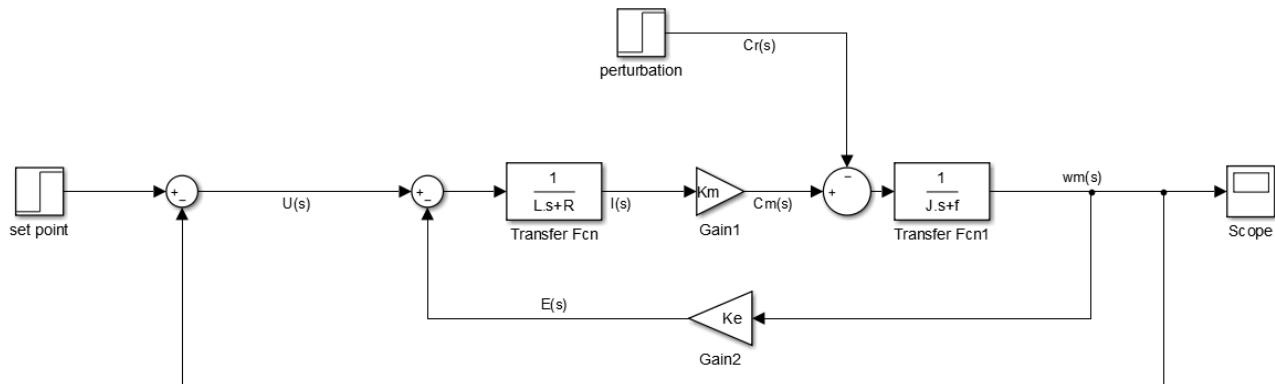


Figure 38: example of Simulink modeling

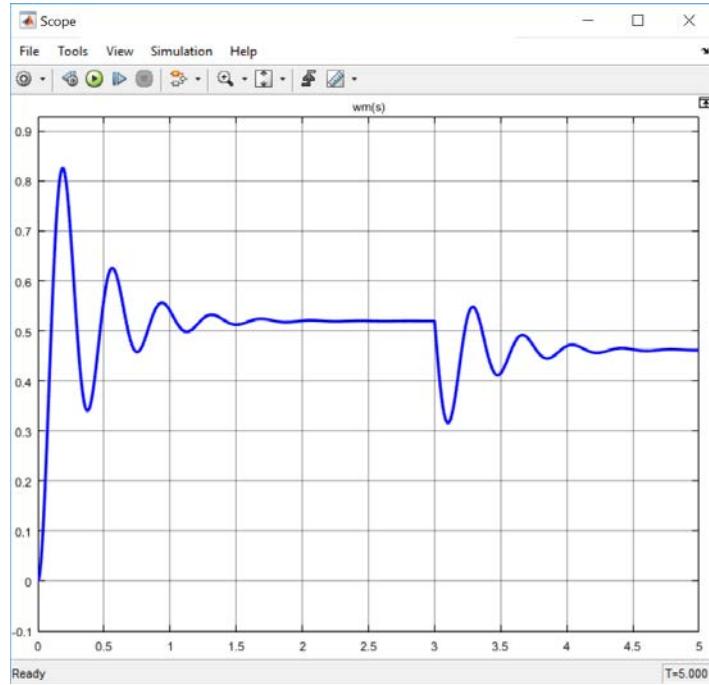


Figure 39: obtained results visualized in a scope

3. Simscape

Simscape offers an acausal approach to modeling and facilitates modeling by assembling components. The physical behavior of the components is taken directly into consideration by the software, such that it is therefore possible to model a system without having to write the differential equation which characterizes its behavior.

Each physical field is represented by a color. Connections between two components in the same domain are not directed, have a physical meaning and transmit a higher level of information than connections in causal modeling. Connections may be an electric wire (transfer of current and voltage information), a drive shaft (transfer of angular coupling and speed information), the end of the cylinder rod (transfer of force and linear speed information) ...The calculation principle is supported by a power balance at each node of the model and is not based on the principle of causality, giving it the name acausal modeling.

Figure 40 shows an example of a model completed with **Simscape** and Figure 41 shows the results obtained.

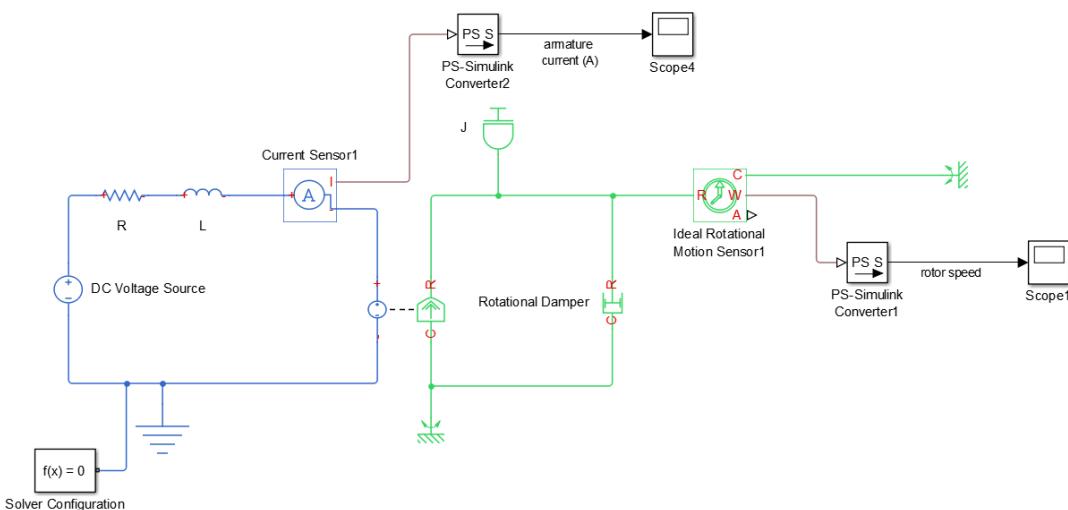


Figure 40: example of Simscape modeling

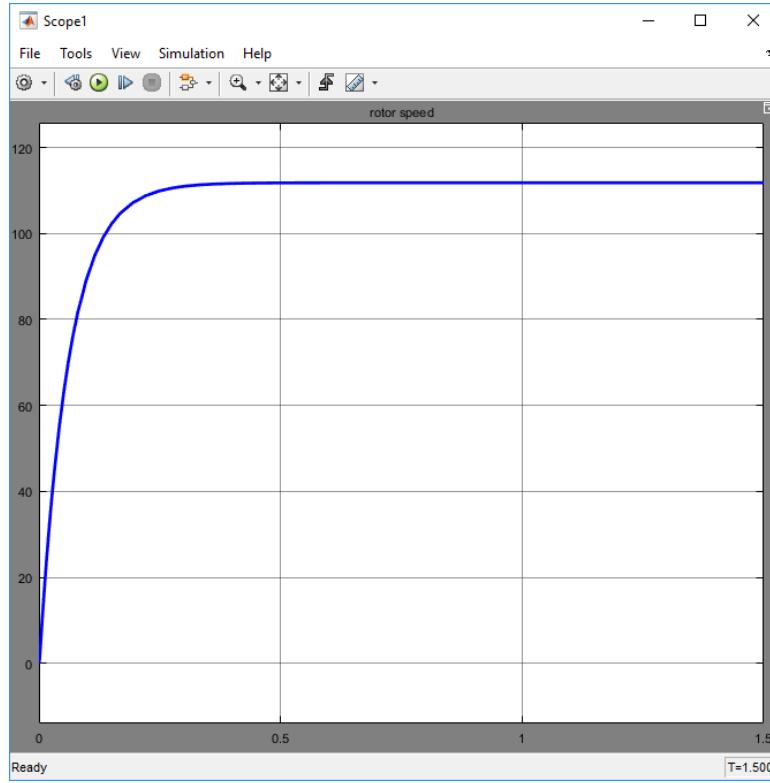


Figure 41: obtained results visualized in a scope

Simscape includes a library of elementary components of all technologies (mechanical, electrical, hydraulic, pneumatic, thermal, magnetic...) and also includes different modules:

- **SimMechanics:** this module enables 3D CAD models to be imported, and for their direct integration into a multi-physics model, a servo controller... The kinetic properties of the parts are taken into consideration and the dynamic behavior is automatically integrated.
- **SimPowerSystems:** electrical components (all technologies for motors, power supply, filters, electric pre-actuators...)
- **SimElectronics:** components for electronics and mechatronics (servo-motors, PWM command, H bridge, convertors, logic ports...)
- **SimHydraulics:** components facilitating the creation of hydraulic circuits (all technologies for pumps, distributors, cylinders, valves...)
- **SimDrivelines:** power transmission components (reducers, brakes, clutches, couplings...)

Each of these modules explores a specific technological field and offers superior modeling to that of the libraries in the **Simscape** database.

Simscape libraries

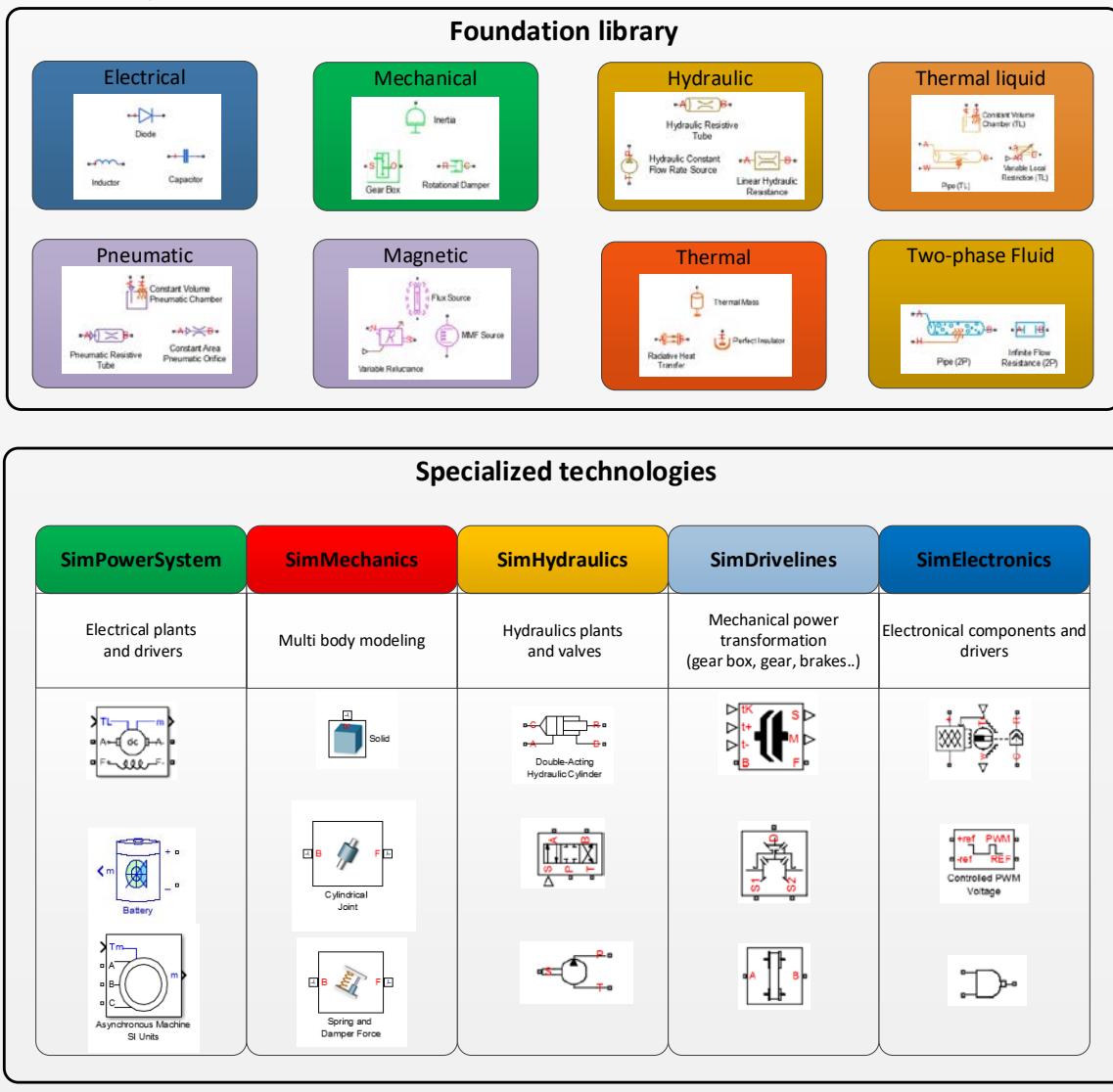


Figure 42: Simscape and its libraries

4. Stateflow

Stateflow controls modeling of the combinatory and sequential behavior of systems using a description in the form of state diagrams, logic flow diagrams or truth tables. Figure 43 gives an example of modeling state machines with **Stateflow**.

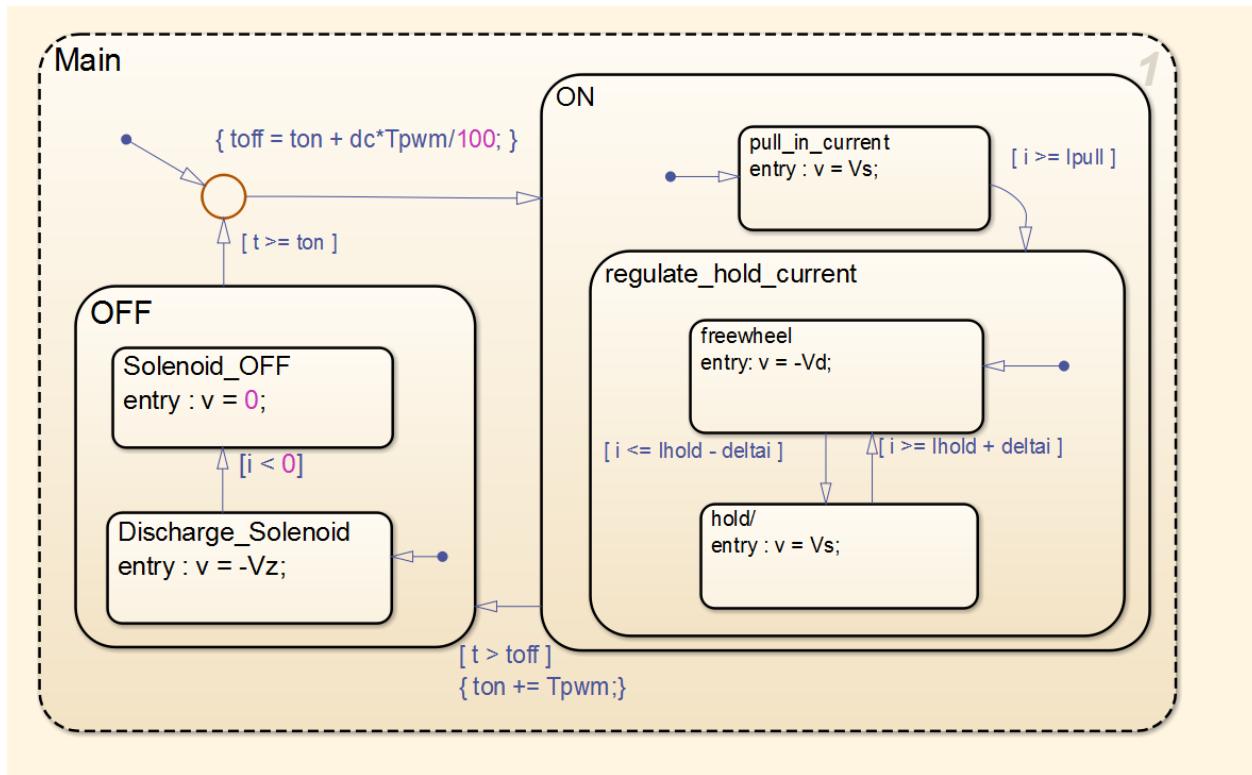


Figure 43: modeling of the behavior of an automatic transmission with Stateflow

5. Using modeling tools

All of these modeling tools communicate with each other and enable the use of a global system modeling procedure. In the same model it is possible to combine causal and acausal modeling, to integrate a CAO 3D model with **SimMechanics**, and to account for a sequential behavior with **Stateflow** all whilst using command controls tools through **Simulink**. It is possible to have the model communicate with **MATLAB** and to use the export results function, to launch a simulation using a **MATLAB** program by automatically varying the desired parameters...

Figure 42 and Figure 44 describe the hierarchical organization of the palette of tools selected in the **MATLAB-Simulink environment**.

First, we'll examine the software environment to become familiarize ourselves with the methods for launching various tools. Then, we will begin to use the tools using examples that may be covered with students as part of engineer training. Performance in the modeling process is based, above all, on knowledge of the foundations of each tool. It will then be easy for the user to make more in-depth use of these tools in accordance with the needs of the modeling process.

MATLAB, Simulink

Simscape

SimPowerSystem

SimMechanics

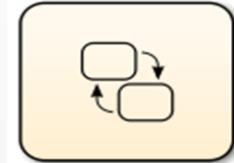
SimHydraulics

SimDrivelines

SimElectronics

Stateflow

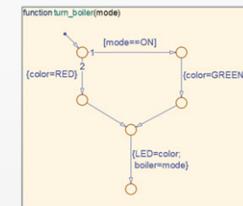
Stateflow



States machine



Truth table



Logical flow

Figure 44: MATLAB for multi-physics modeling

II. Presentation of the MATLAB – Simulink environment

A. Launching the software



Launch the MATLAB software using the **MATLAB R2015b** icon located on the desktop.

B. The MATLAB environment window

When you open MATLAB, more windows are accessible.

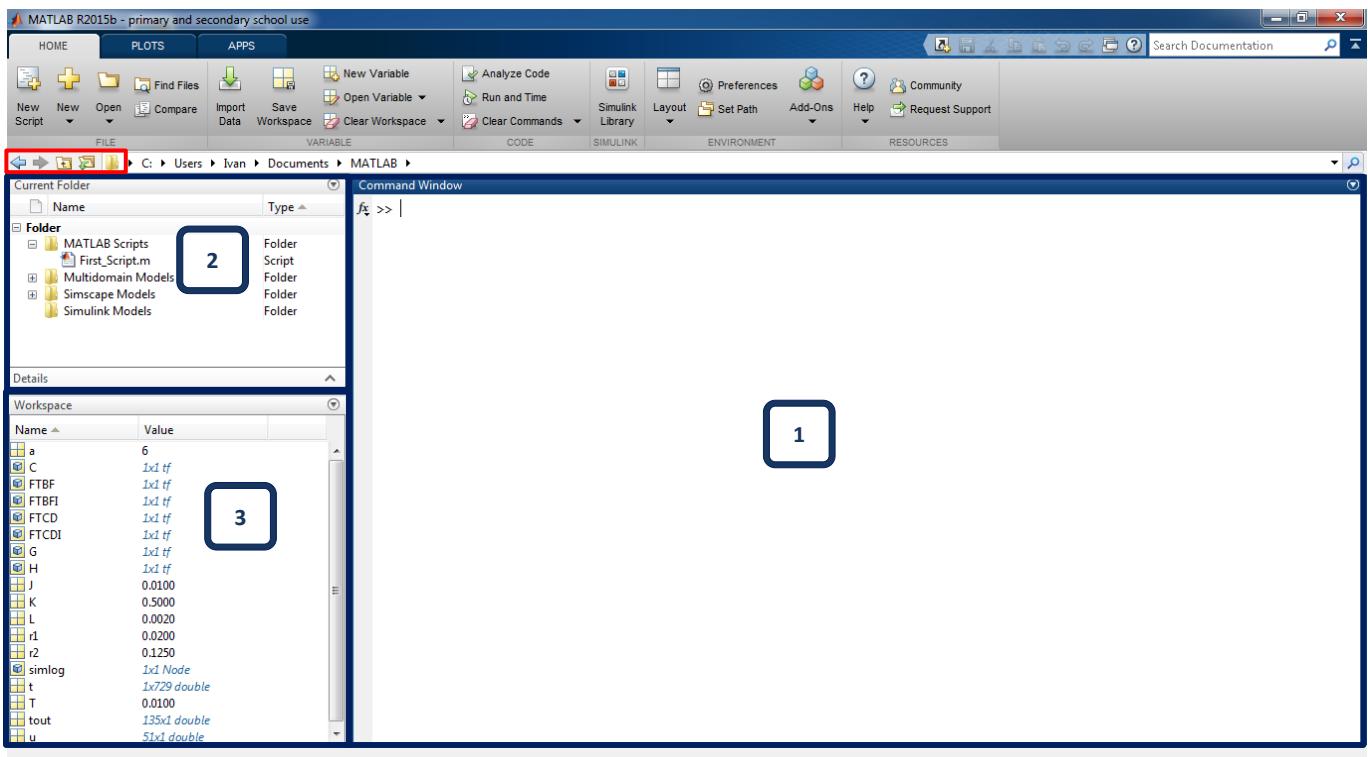


Figure 45: MATLAB environment window

1: Command Windows

MATLAB command window This is the window through which the user communicates with MATLAB (inputting command lines, instructions...)

2: Current Folder

This window indicates the current file currently being accessed by the software.

The tool bar



located in the upper portion of this window is used to navigate

the files.

3: Workspace

This window indicates the name and format of all variables created and used during the current session. It is possible to save the Workspace in a file.

1. The MATLAB command bar

Figure 46 shows the principal commands on the MATLAB command bar.

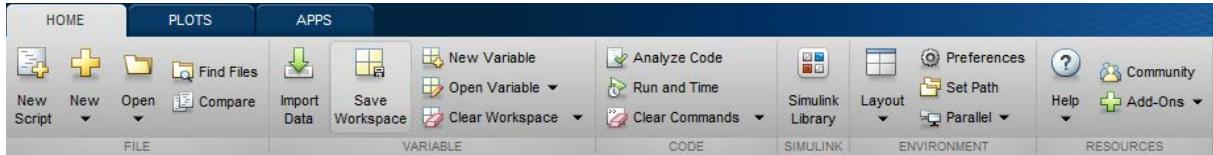


Figure 46: MATLAB environment command bar

C. The Simulink environment window.

To launch Simulink, click on the icon



in order to open the Simulink component library.

The Simulink library opens (Figure 47) and offers a palette of components for causal or acausal modeling.

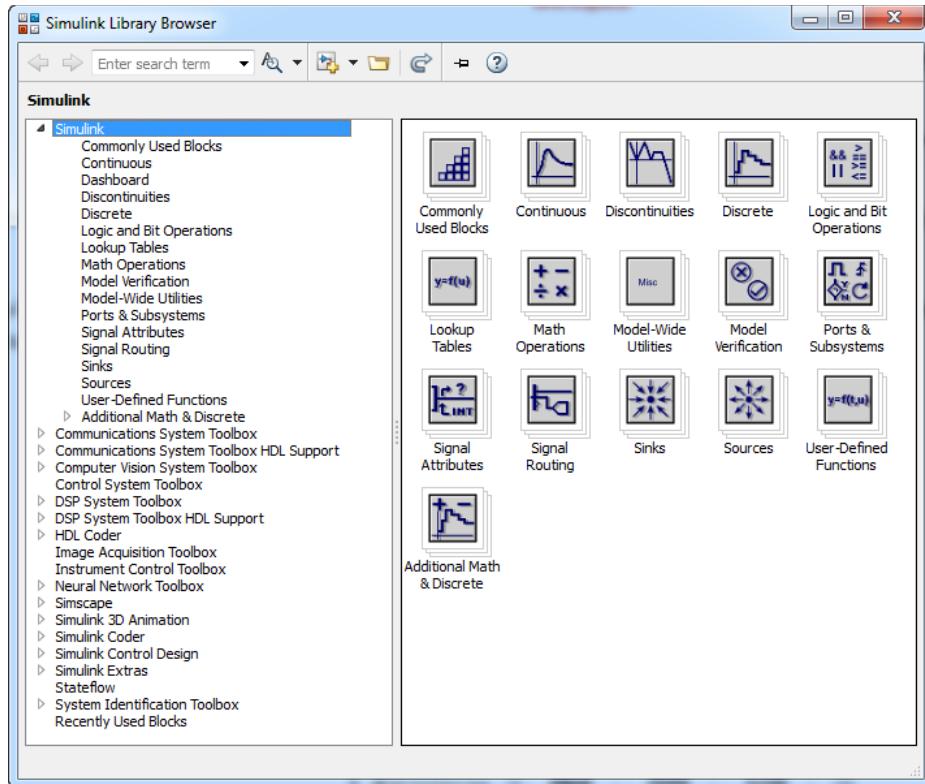


Figure 47: Simulink library

Click on the icon to create a new Simulink library.



The **Simulink** environment window will open. Figure 48 shows the principal commands that can be accessed from this window.

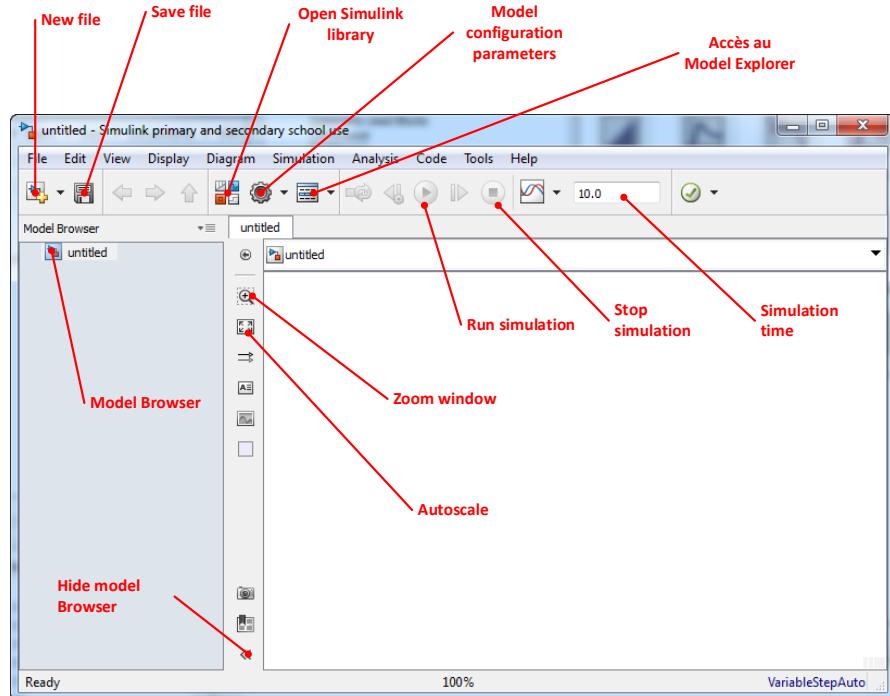


Figure 48: Simulink environment window

Components can be placed by simply dragging and dropping the components from the **Simulink** library to this window.

D. MATLAB – Simulink Configuration

1. Naming a file in MATLAB/Simulink

You can give the name of your choice to a file in MATLAB/Simulink as long as the following rules are followed:

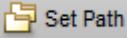
- Do not use spaces
- Do not use accented characters

2. The MATLAB “path”

The **MATLAB** “path” is constituted of all of the files that **MATLAB** can read during one current session. Any file belonging to one of the **MATLAB** “path” files can be accessed without indicating its path. All of the files that contain our working files should thus be included in this “path”. This strategy for accessing files enables us to avoid the need to specify the access path for functions and files. As long as these are able to be found in the “path” specified by the user, this is sufficient for them to be automatically retrieved. The user will type the name of a script or call up a function in the command window for immediate execution. This implies, for example, that two scripts may not have the same name.

It is therefore important to master the structure of the **MATLAB** “path” and to include only the files needed for the working session. There are two procedures for adding files to the “Path.” One adds the files selected for the current working session and for all future sessions. The other adds selected files only for the current working session. It is important to have a thorough understanding of this difference and to include definitively in the “path” only those files that are truly useful for all sessions in order to avoid possible conflicts between files with the same name.

3. Adding files to the “path” for all sessions

In order to add files to the **MATLAB** “path” for the current session and for all future sessions, click on  in the command bar to open the “path” definition window.

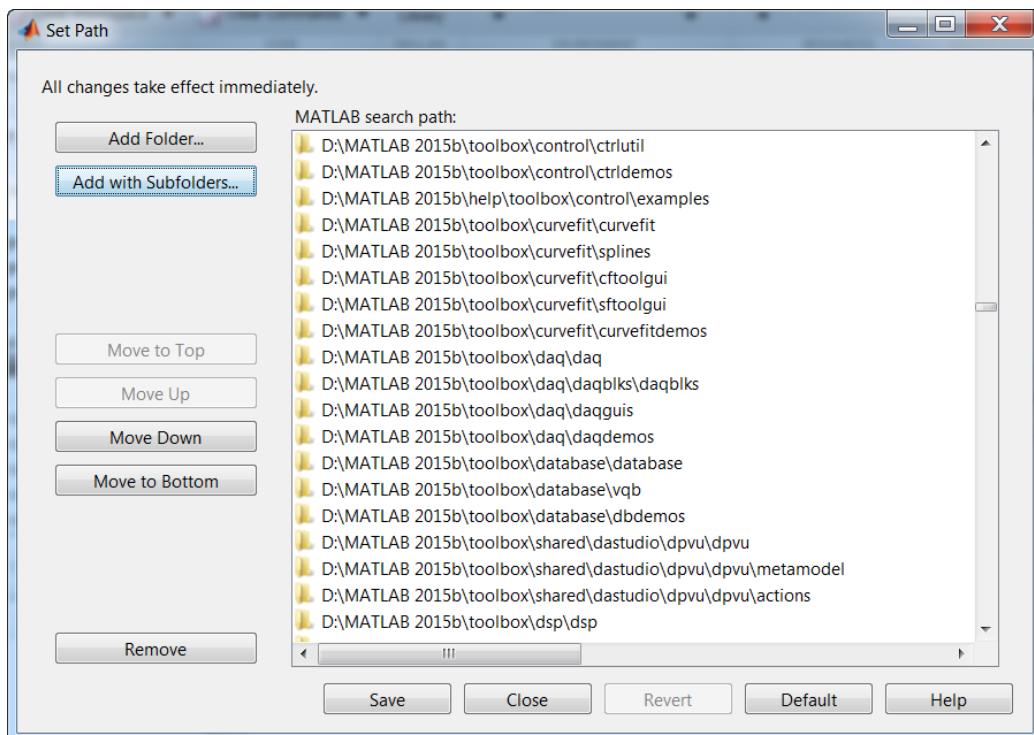


Figure 49: the MATLAB path

Select **Add with Subfolders**, then choose the files you want to add to the “path” of the software (subfolders will be added automatically)

Select **Save**, then **Close**.

Following this operation, the “Path” is saved and will be the same when opened in following sessions.

4. Add files to the “path” for the current session

The “Path” may also be modified for the current session only. To do this, right-click on the file that you want to add to the “Path” in the “Current Folder” window (Figure 50) to display the contextual menu. Then select **Add to Path** then **Selected Folders and Subfolders**.

It is also possible to use this procedure to delete files from the “Path” by selecting **Remove from Path** then **Selected Folders and Subfolders**.

In order to be able to use all of the simulation files used in this book, add the **Models_book_2015b_Ivan_LIEBGOTT_US** folder to the “Path” in MATLAB.

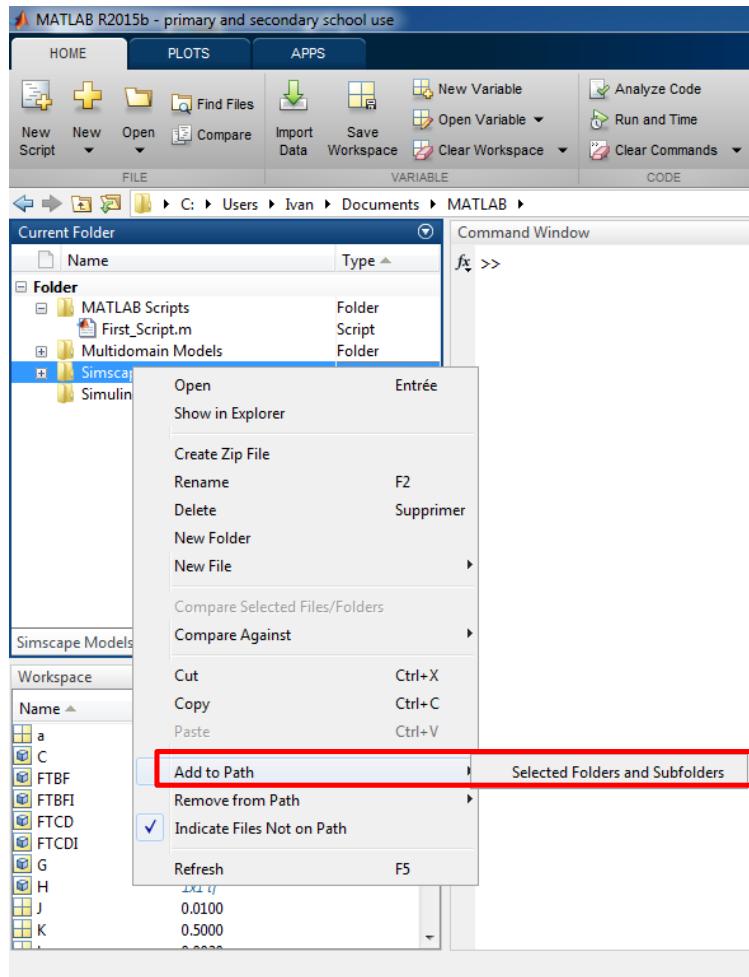


Figure 50: adding a file to the “Path” for the current session

Files that do not belong to the “Path” are grayed-out in the **Current Folder** window of the **MATLAB environment**.

III. Design strategy for a Multi-Physics Model

A. Link to the Energy chain/information chain diagram

The energy chain/information chain descriptor is especially suited to a multi-physics modeling approach. It enables the display of the energy transformation structure in the system and its interactions with the command device. In addition, it is possible to make a complete analogy between this descriptor and the model executed with **MATLAB-Simulink**.

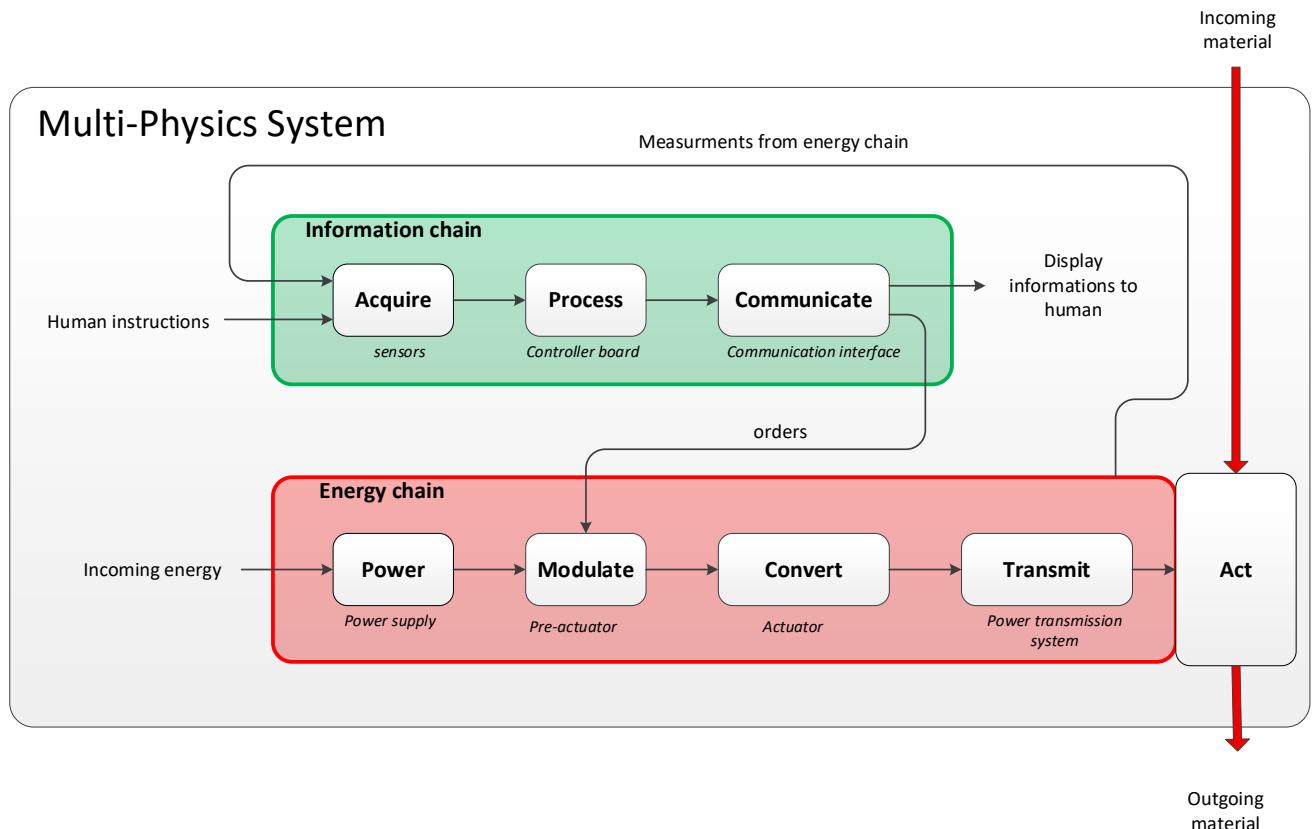


Figure 51: Energy Chain/Information chain diagram

In order to implement a multi-physics modeling strategy, the strategy can be based on this descriptor by connecting the parts of this diagram to the selected modeling tool. A trend must be identified, since each modeling problem is unique and numerous strategies can be used.

It should also be possible to use a descriptor from the **SysML** language, such as the internal blocks diagram, which enables using the same type of approach and level of description as the energy chain / information chain diagram.

The diagram in Figure 52 illustrates this procedure.

MATLAB-Simulink

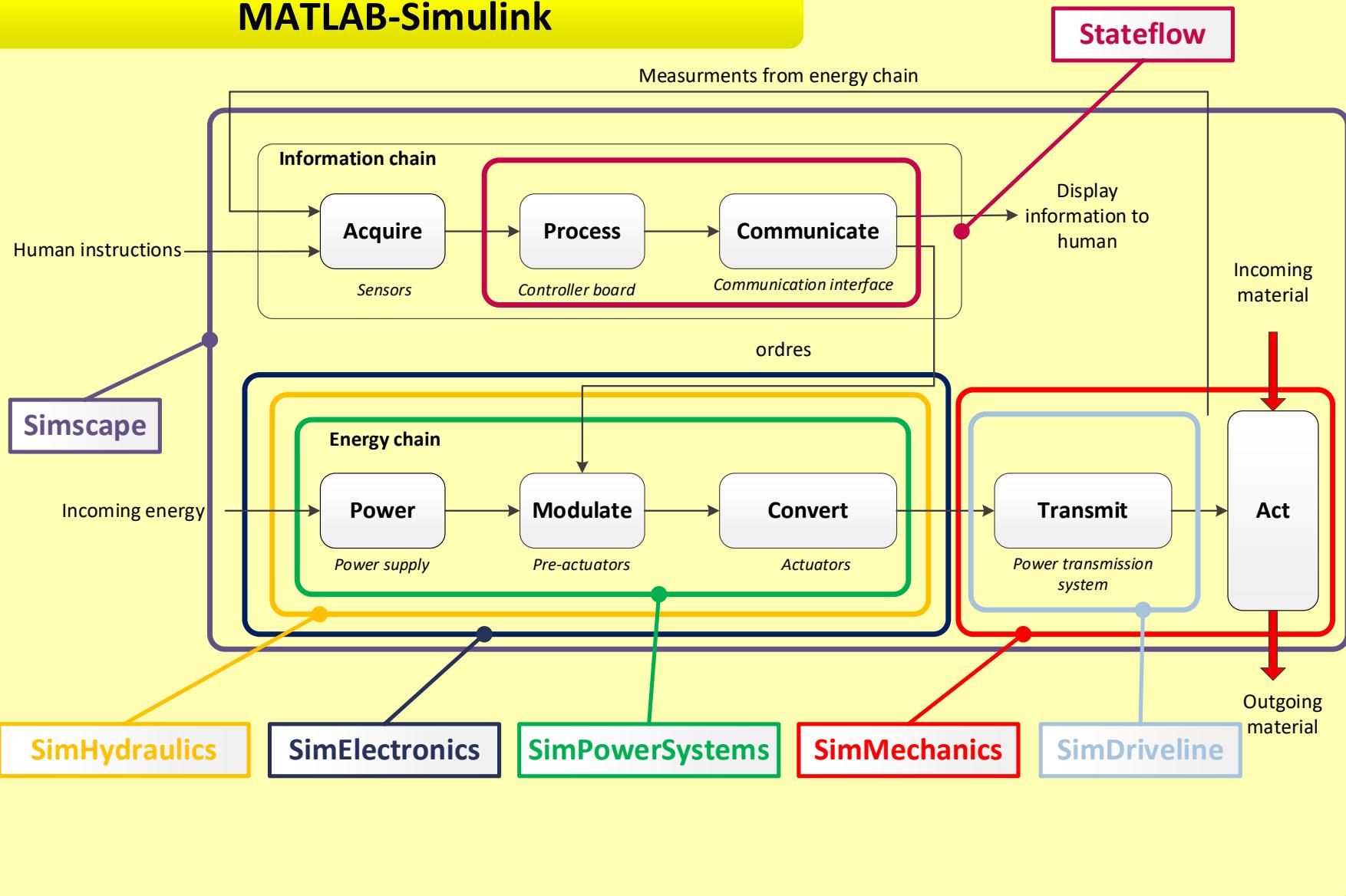


Figure 52: correspondence between the energy chain / information chain diagram and MATLAB tools

IV. Application to a hydraulic boat pilot

In order to illustrate this approach, we will apply the proposed procedure to the automatic hydraulic pilot of a boat. The system functions as follows. A DC motor operates a double-flow hydraulic pump which flows into a cylinder through a hydraulic distribution circuit. Transfer from the cylinder involves the rotation of the stock arm on which the boat's helm is fixed. Rotation of the helm modifies of the course of the ship. The information chain controls the motor by ensuring that the real course of the boat follows the course instruction.

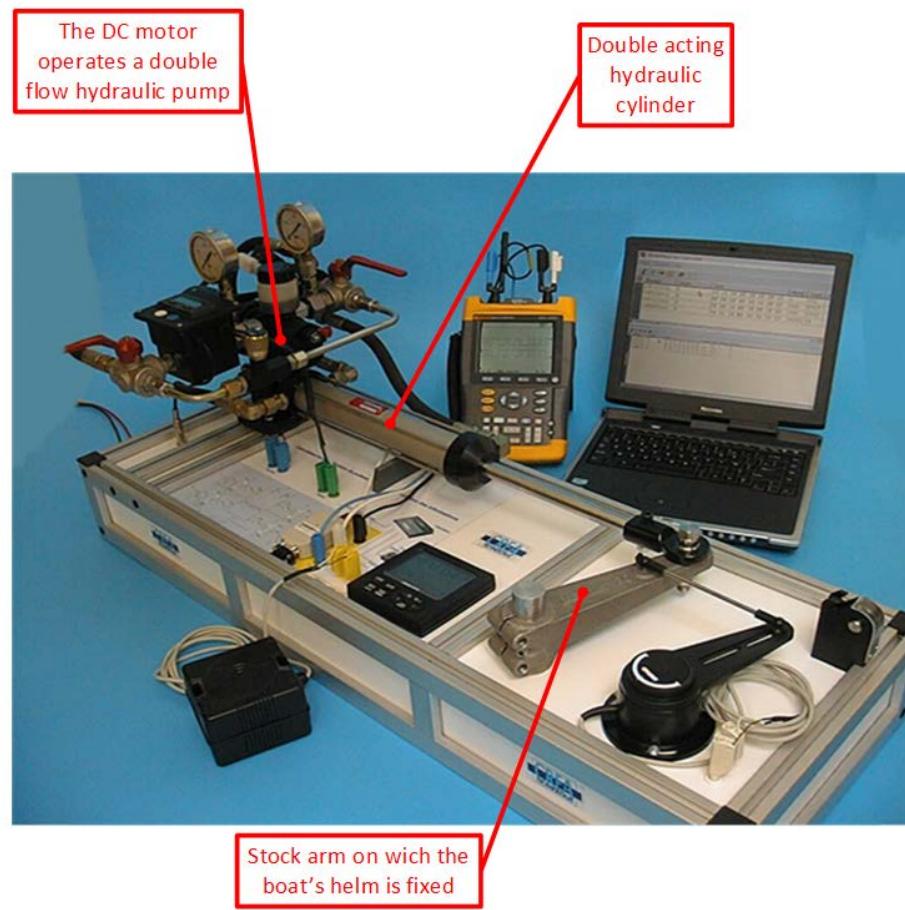


Figure 53: photo of the automatic pilot

A. Diagram showing the energy chain and information chain of the boat hydraulic pilot.

Figure 54 shows the energy chain/information chain of the boat pilot and the tools chosen to model the different parts of the system.

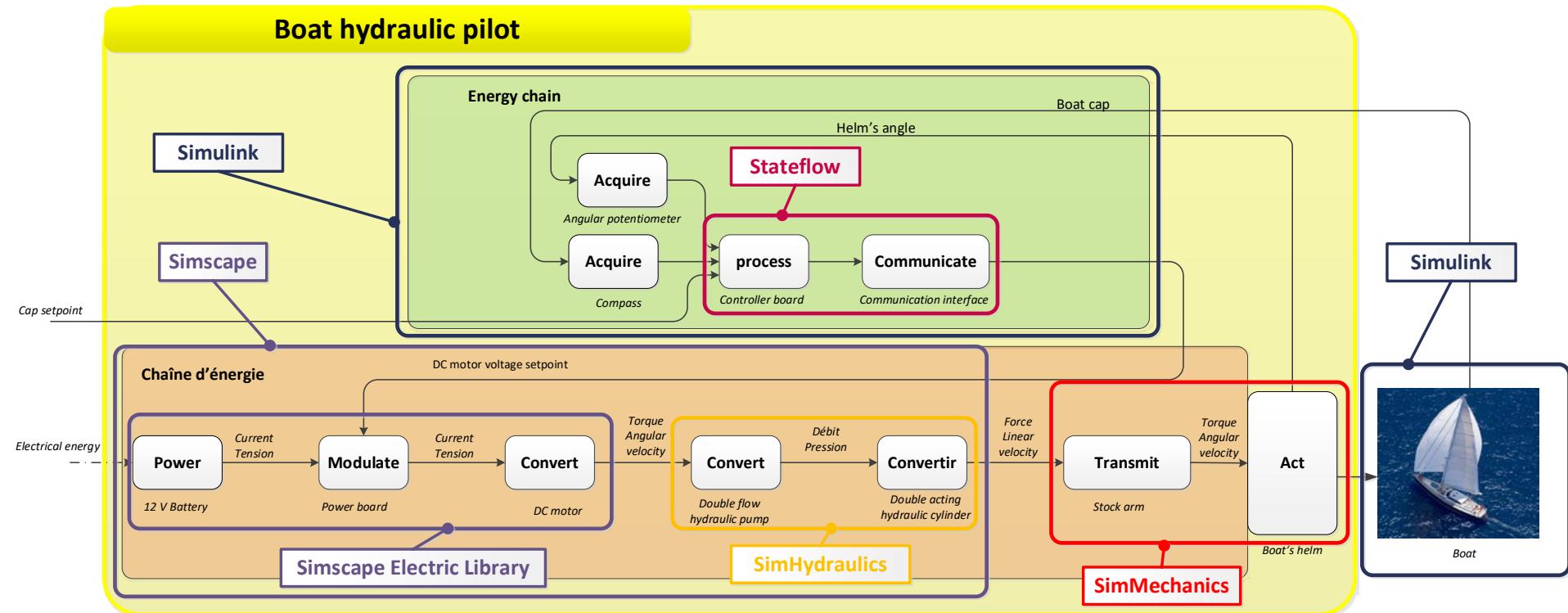


Figure 54: relationship between the energy chain/information chain diagram and the MATLAB tools

B. Multi-physics model of the boat hydraulic pilot executed with MATLAB - Simulink.

The multi-physics model is structured by following exactly the same formalism as the descriptor. The information and energy chains appear as subsystems. Exchanges of energy and information between the blocks of the descriptor are the same as the connections between the subsystems of the multi-physics model. Modeling can be accessed by double-clicking on a sub-system. Multi-physics modeling also becomes a helpful tool for description which enables the dynamic exploration of the components that execute the functions.

Figure 55 shows a copy of a screen for a multi-physics model completed with MATLAB – Simulink.

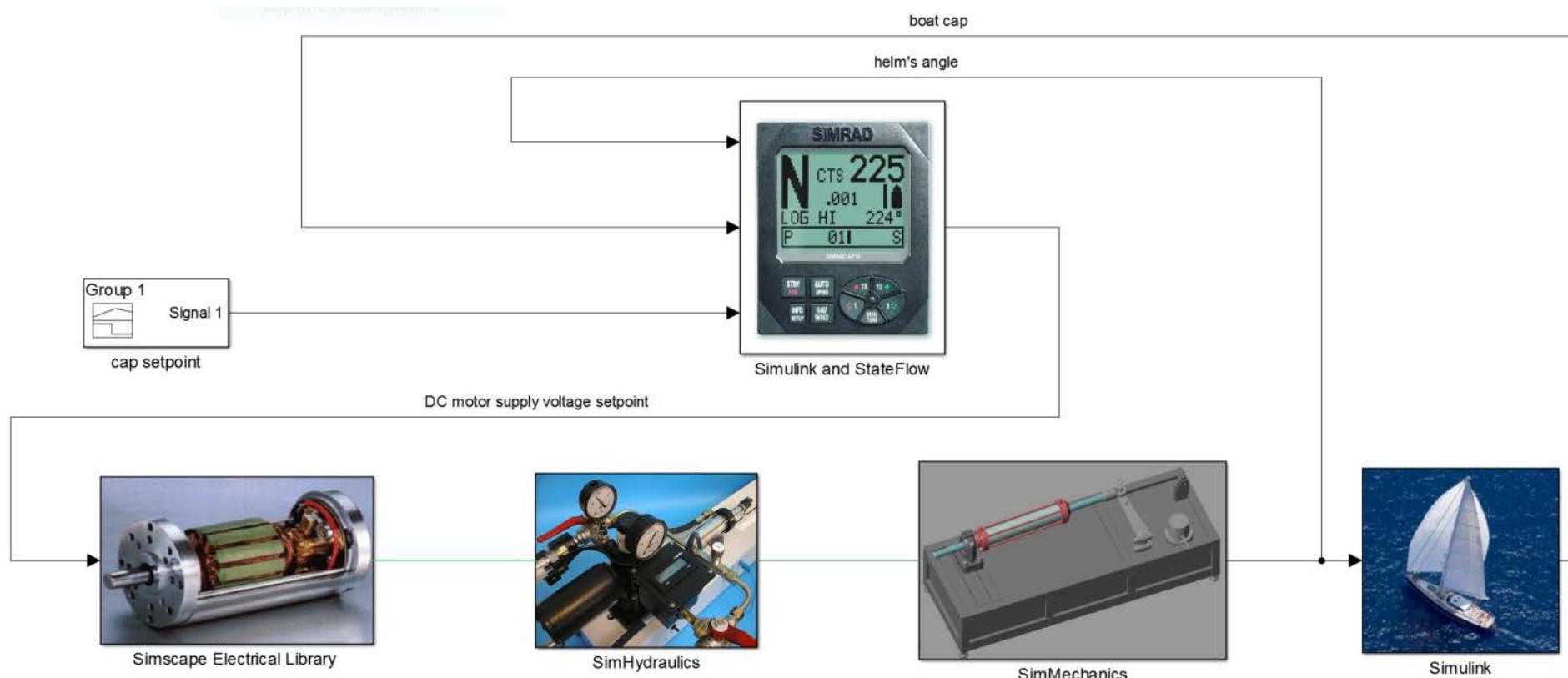


Figure 55: copy of the multi-physics modeling screen of the boat automatic pilot executed with MATLAB

C. Model loading and simulation

In order to be able to use all of the simulation files used in the work, the **Models_book_2015b_Ivan_LIEBGOTT_US** folder must be included in the MATLAB “Path” (procedure described in 42).

D. Visualization of the results of the multi-physics model

Open the **pilot_hydraulic.slx** file (*Chapitre_1_Introduction/Pilote_hydraulique*)

Double-click on the “consigne de cap” (course instruction) block to display the instruction signal. The course instruction directs a course of 20° for 300 s then a course of -40° for 700 s.

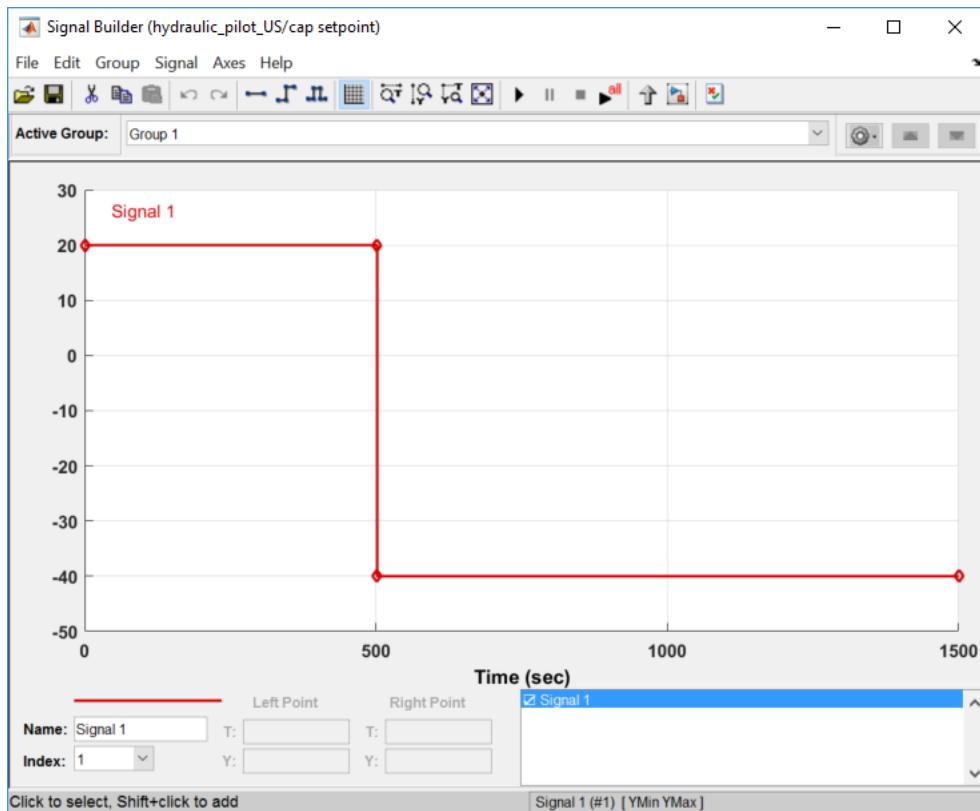


Figure 56: View of the course instruction



Launch the simulation by clicking on

When the simulation launches, a **Mechanics Explorers** window opens to allow dynamic visualization of the movements of the mechanical section, modeled using **SimMechanics** (Figure 57).

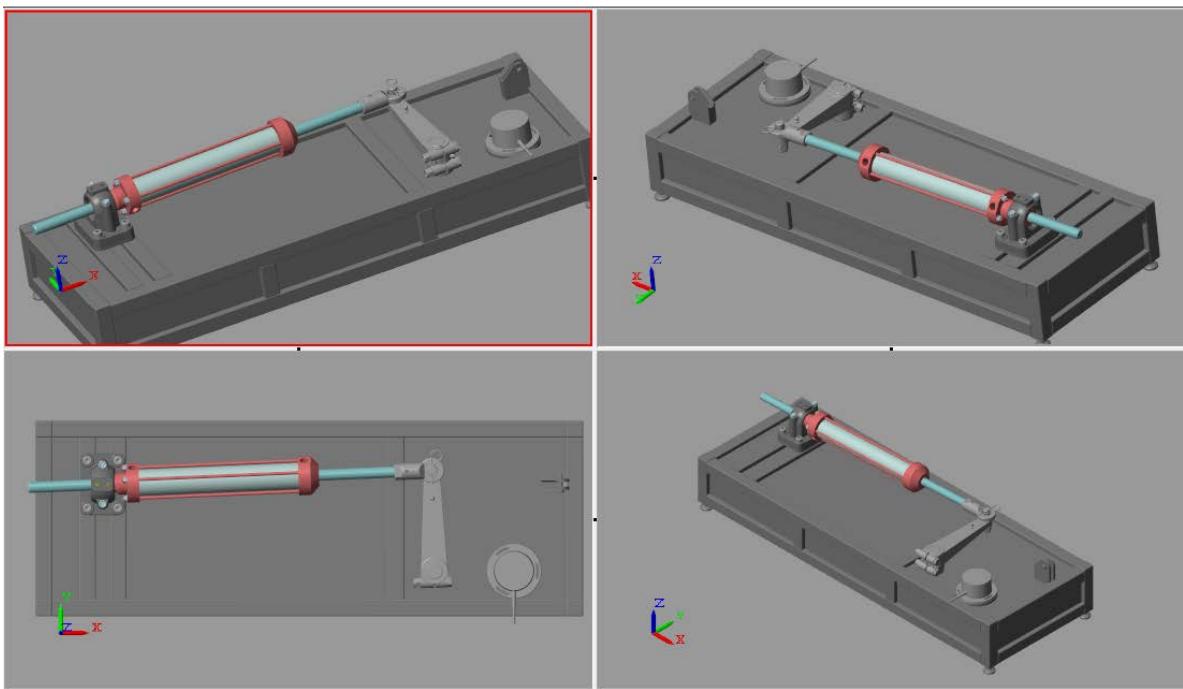


Figure 57: SimMechanics visualization window

A visualization bar allows you to choose the observation mode for the model.

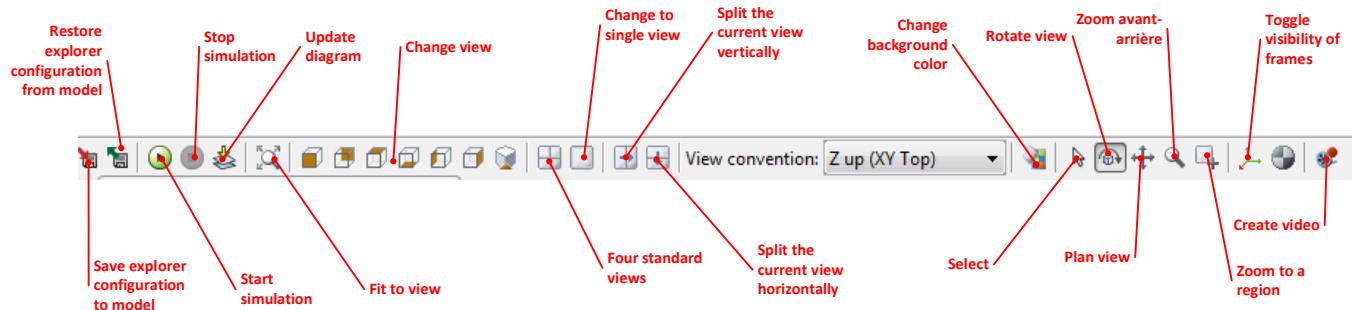


Figure 58: SimMechanics visualization bar

Click on the icon in the upper part of the **Mechanics Explorers** window (Figure 58) to return to a display with a single window (Figure 59).

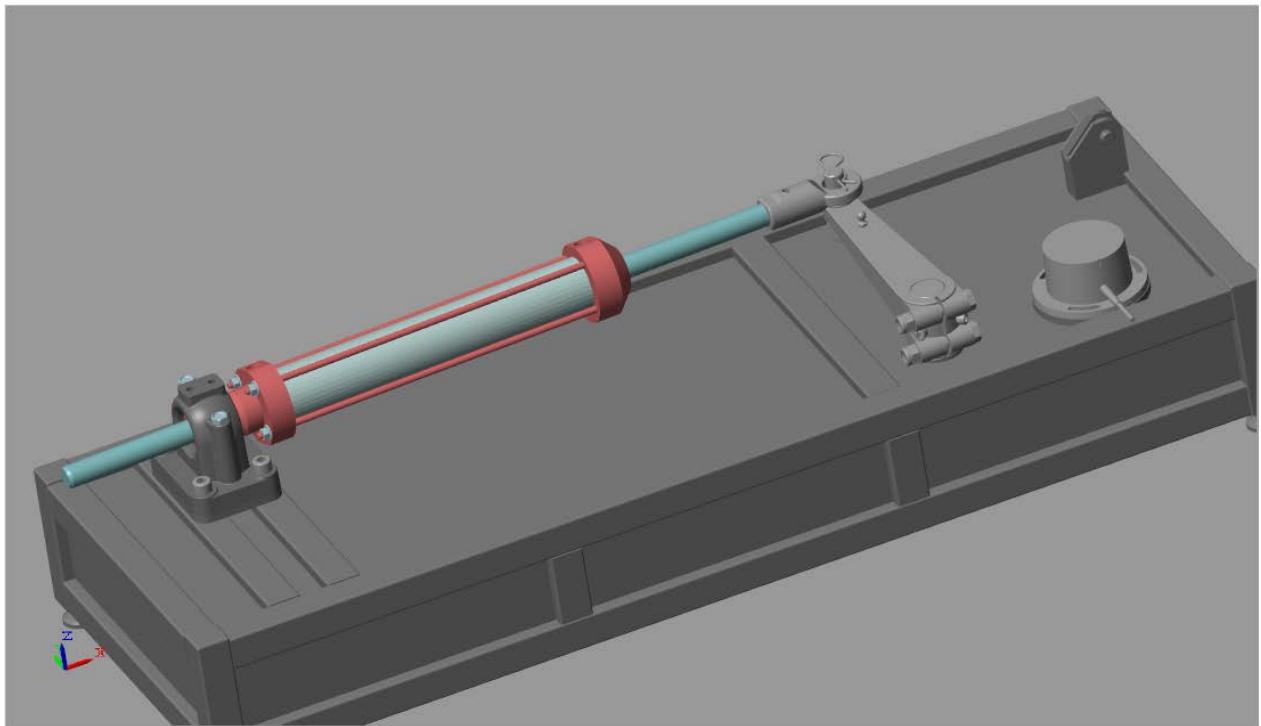


Figure 59: SimMechanics visualization window

In the lower portion of the window, a time bar enables changes in the model over time to be visualized. Visualization can be accelerated using a cursor or returning to any instant in the simulation(Figure 60).



Figure 60: SimMechanics time bar

At the end of the simulation, the results can be easily used by double-clicking on the scopes. All the physical dimensions of the model can be enlarged.

We can visualize the course instruction and the effective course followed by the boat:

Click on the automatic scaler to visualize the entire curve.



Try to develop the habit of using the automatic scaling command when opening each scope to visualize the signal.

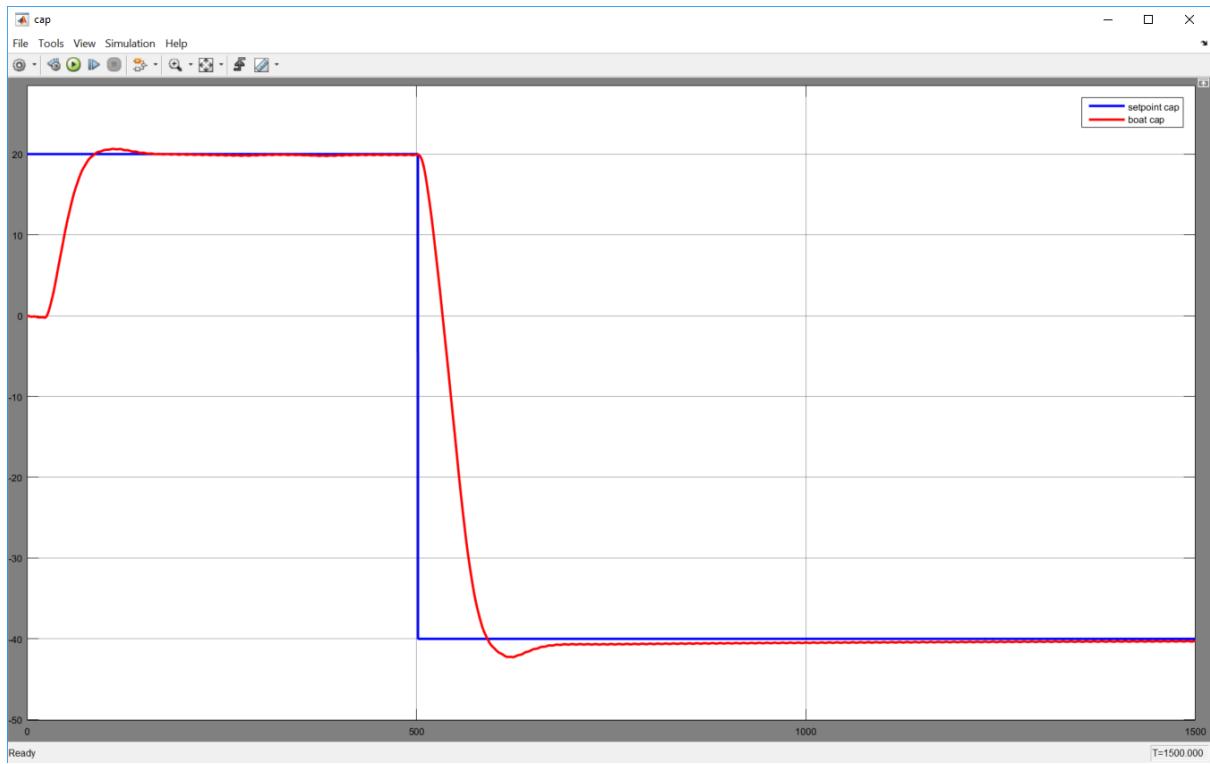


Figure 61: View of the course instruction and of the effective course followed by the boat

Current, voltage and torque of the motor:

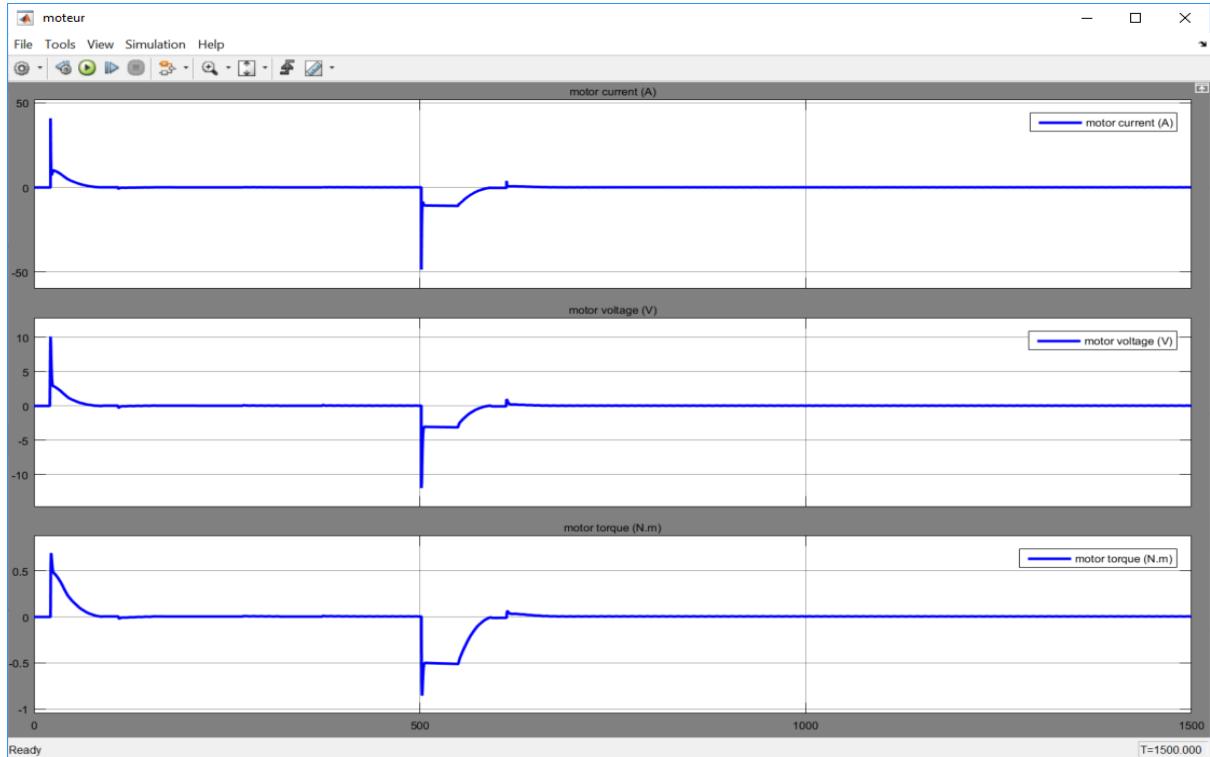


Figure 62: View of the relative dimensions of the motor

Rotation angle of the helm:

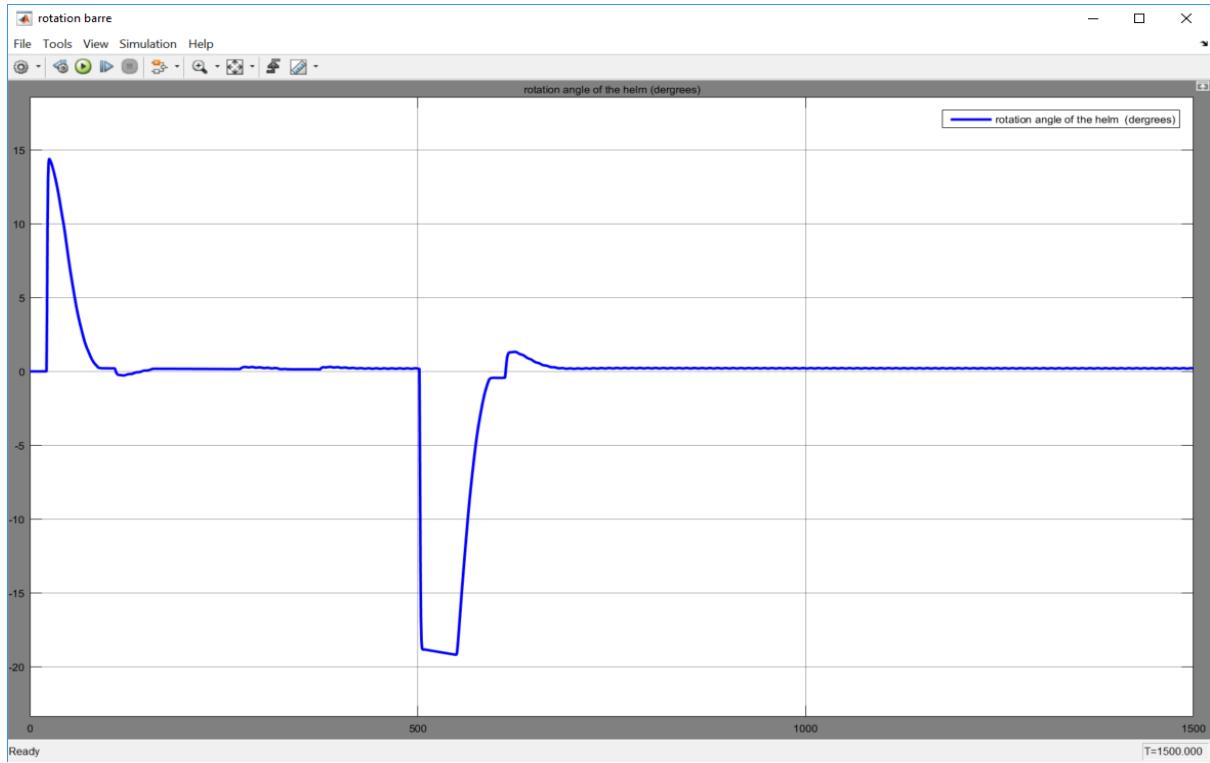


Figure 63: View of the rotation angle of the helm

Pressures in the chambers in front and behind the cylinder:

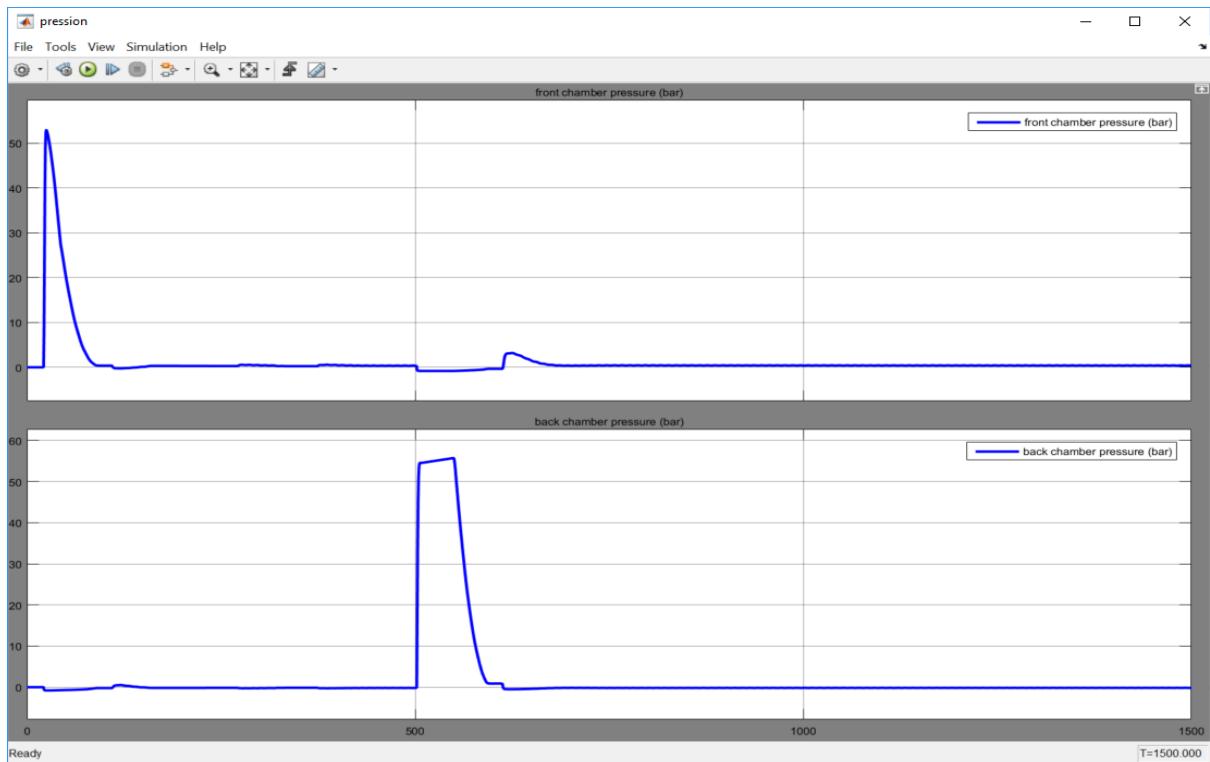


Figure 64: View of pressures in the chambers in front and behind the cylinder

Flow rates in the chambers in front and behind the cylinder

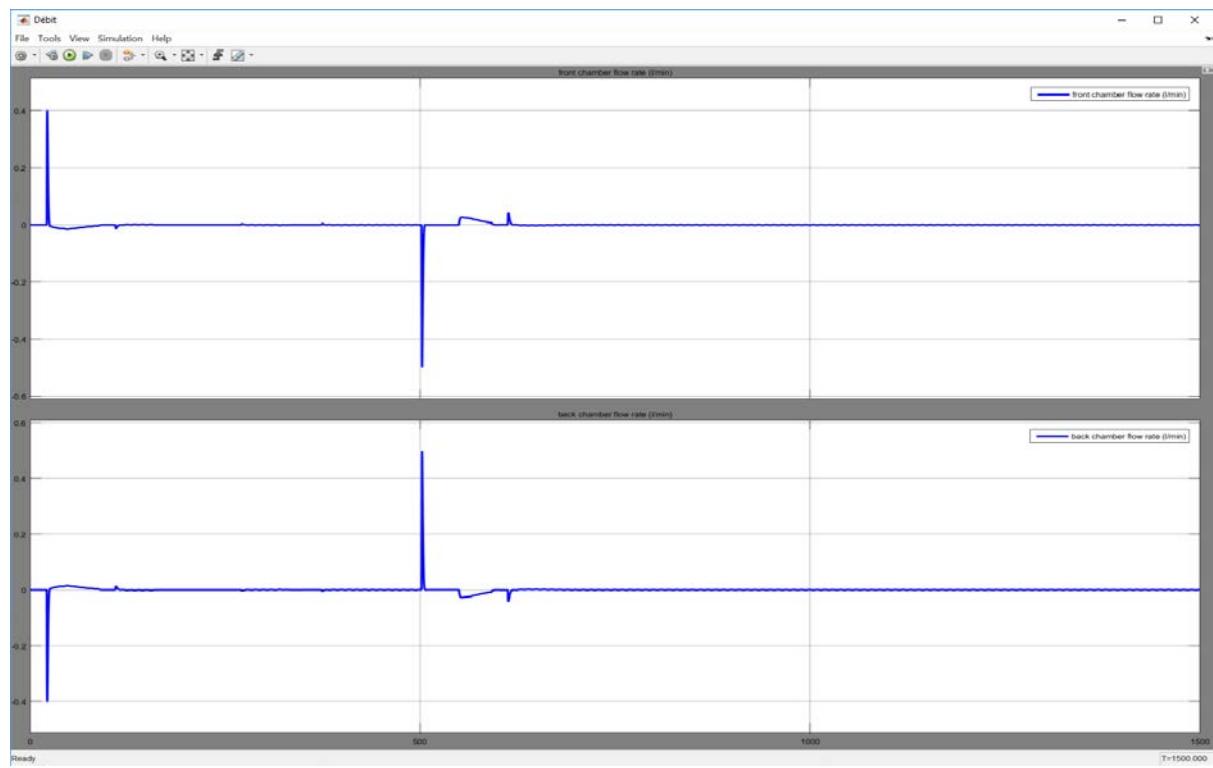


Figure 65: View of the flow rates in the chambers in front and behind the cylinder

E. Exploring the model

1. Exploring the information chain model: Simulink and Stateflow

The information chain is modeled using **Simulink** and **Stateflow**. The functioning of the system requires a loop to servo control the helm into position. This loop is regulated by a PID controller. The second loop allows compliance with the course instruction. The logic is programmed in the form of state diagrams with **Stateflow** and allows the regulation system to act only if the boat leaves its course for more than 20 s.

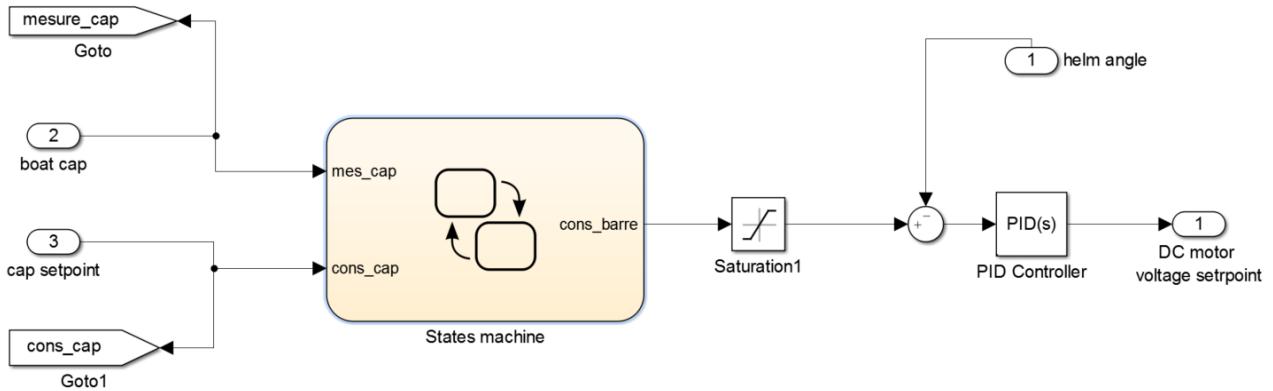


Figure 66: MATLAB – Simulink model of the autopilot information chain

Course loop state diagram:

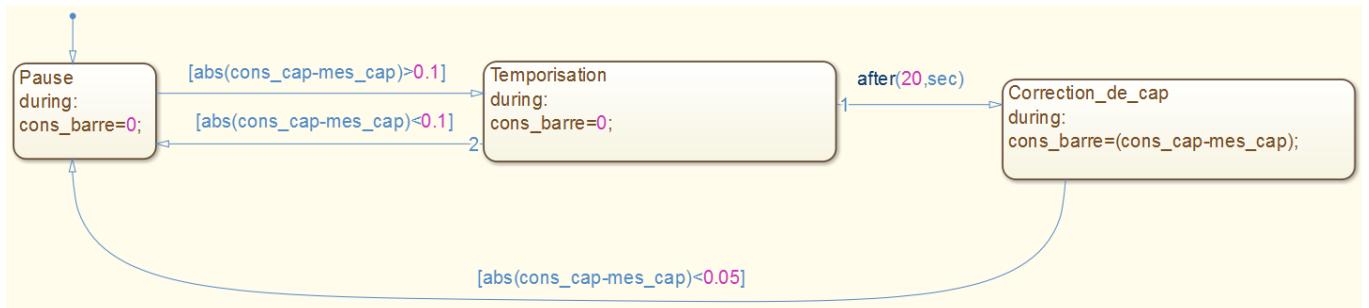


Figure 67: modeling the course control with state diagrams

2. Exploring the energy chain model: Simscape Electric Library

The electric portion of the energy chain is modeled using basic elements from the library of electric components in **Simscape**. The DC motor is modeled using an inductance, a resistance and a component for converting electrical power into mechanical power. Measurements of current and voltage are taken in the electric circuit, one torque measurement is also made on the drive shaft.

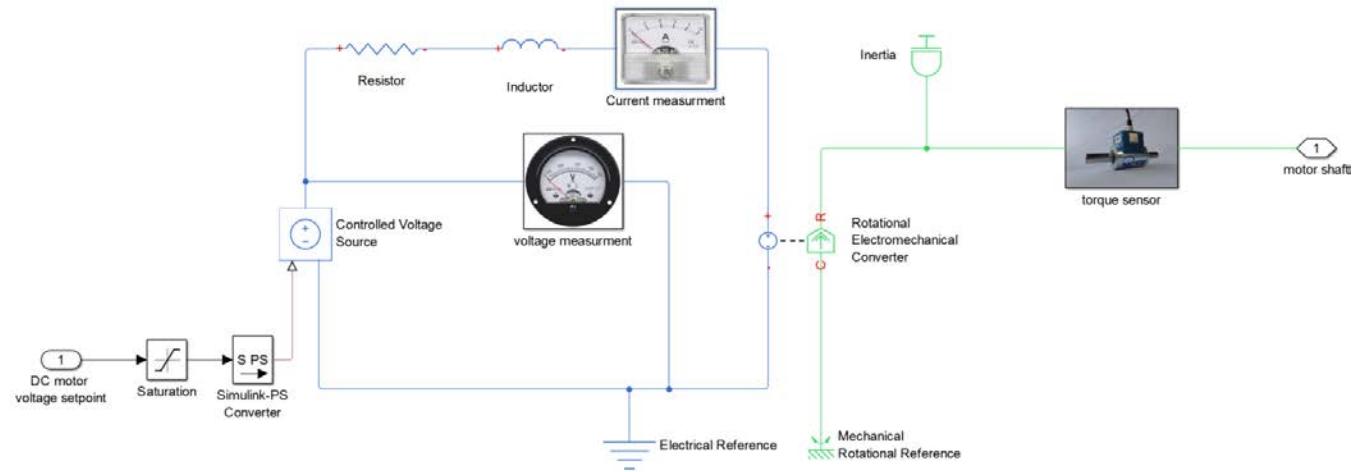


Figure 68: Simscape model of the electrical portion of the autopilot energy chain

3. Exploring the energy chain: SimHydraulics

The hydraulic part of the energy chain is carried out using components from the **SimHydraulics** library. The two-way pump delivers variable displacement flow directly into a double-acting cylinder. Pressure limiters allow the discharge of the fluid when the piston rod is at a stop. Pressure and flow measurements are carried out at different points of the hydraulic circuit.

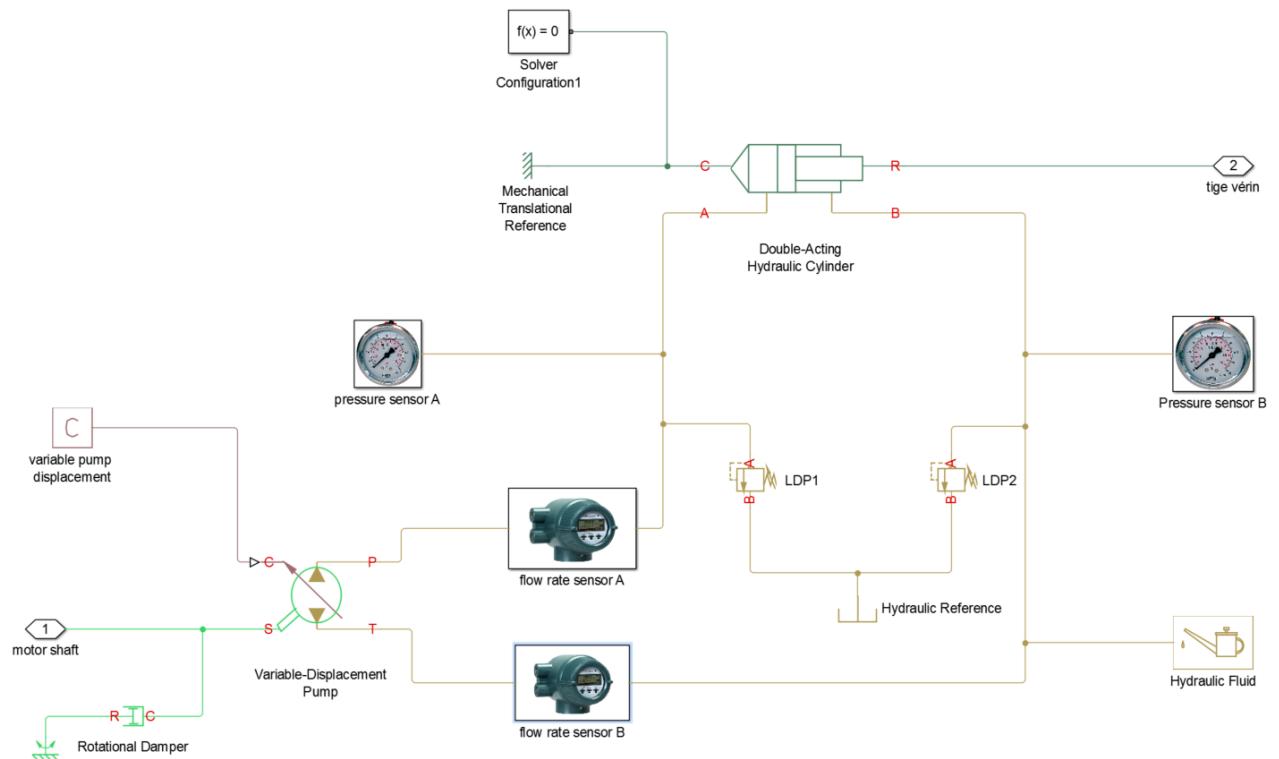


Figure 69: SimHydraulics model of the hydraulic part of the autopilot energy chain

4. Exploring the energy chain: SimMechanics 2G

The mechanic part of the energy chain is created using **SimMechanics** 2nd generation. The model is presented as a Bond graph. It shows the solids and the links between the solids. Masks (images) were applied to the solid to improve the understanding of the model. Displacement action of the piston rod and rotation of the helm take place. A block also defines the nonlinear action of the forces exerted on the rudder by the water. A **SimMechanics** model can be built directly with **MATLAB** or be imported using CAD software such as **Solidworks**.

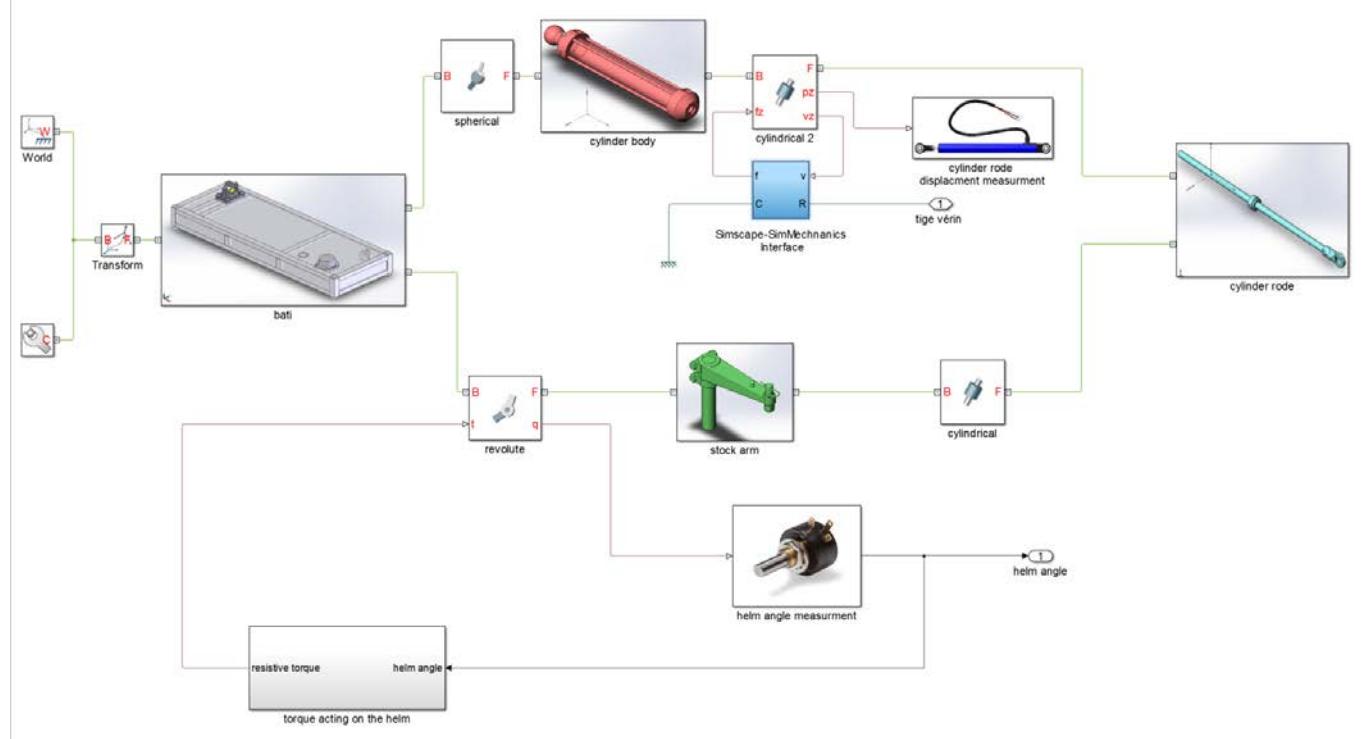


Figure 70: SimMechanics model of the autopilot structure

5. Exploring the energy chain: Simulink

The model enables the dynamics of the boat and the perturbations it undergoes to be taken into account with **Simulink**.

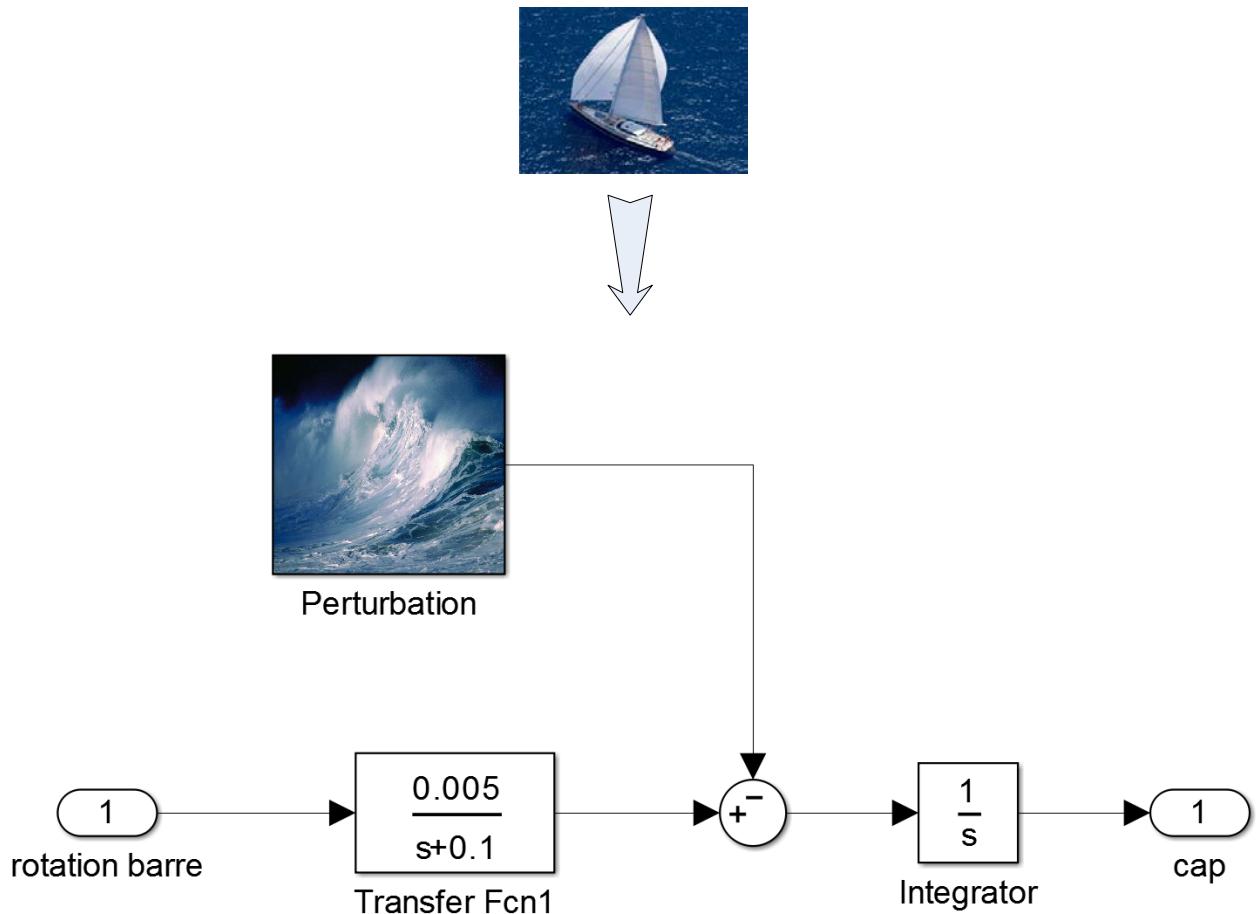
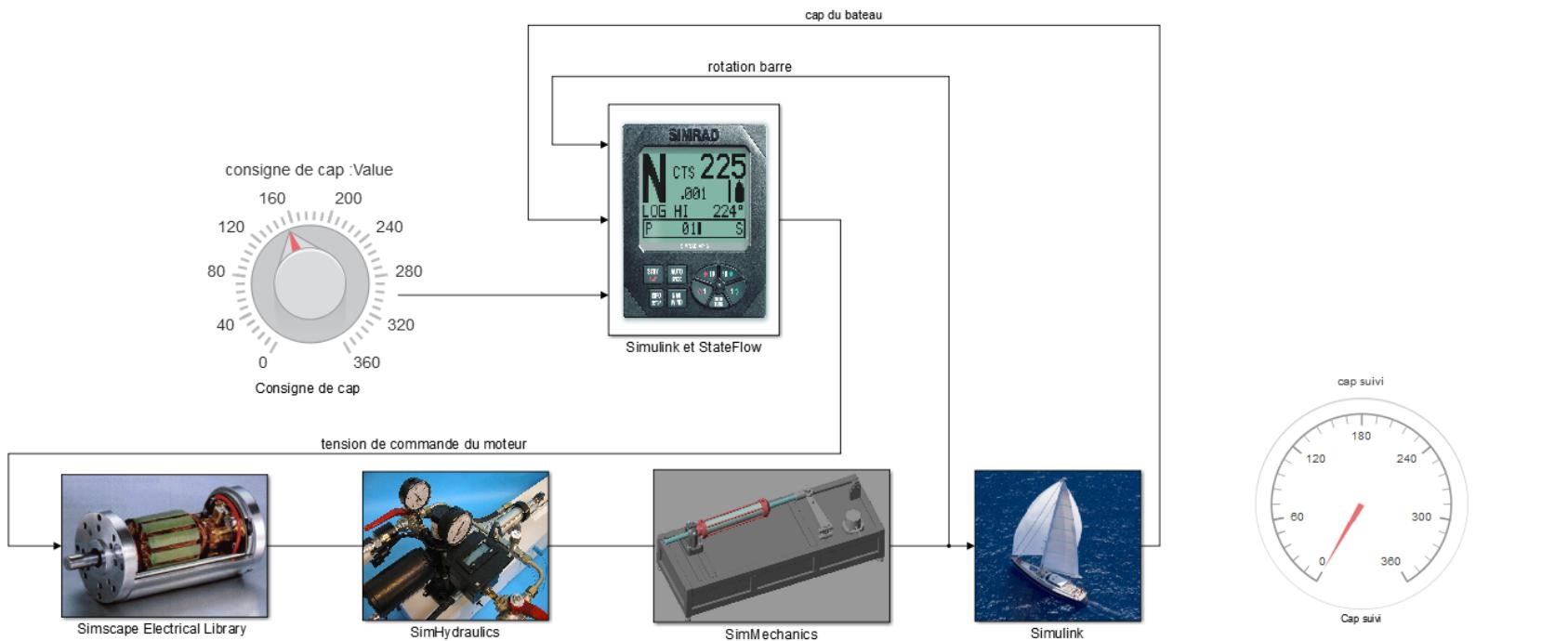


Figure 71: Simulink modeling of dynamic behavior of the boat

F. Interactive piloting of the model

It is also possible to use the Simulink **Dashboard** library to interact with the model during simulation. Its usage and the block configuration are dealt with later in this work (page 156).

Open the “**hydraulic_pilot_dashboard_US.slx**” file. This model is the same as the previous, another point of view has been chosen to illustrate interactive subsystems and settings and display elements have been added. We will see the importance that should be placed on the presentation of the model to make it as accessible as possible for analysis and modification.



Ivan LIEBGOTT @2016

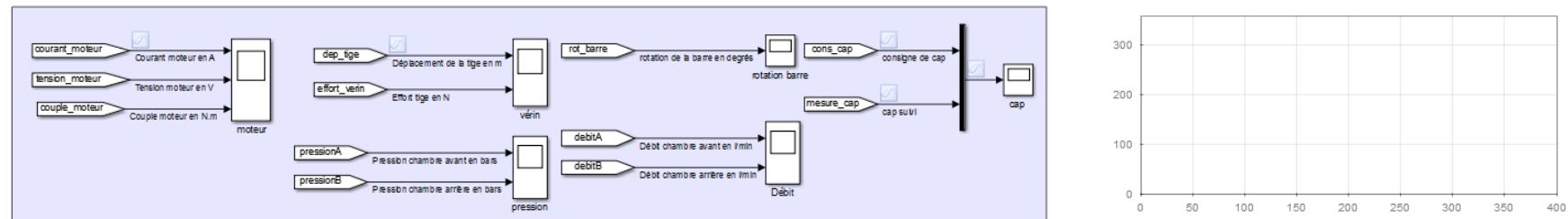


Figure 72: model of the hydraulic pilot with interactive piloting

Launch the simulation and **Modify** the required course by rotating the button during the simulation.

Observe the course in the interactive scope and on the display quadrant.

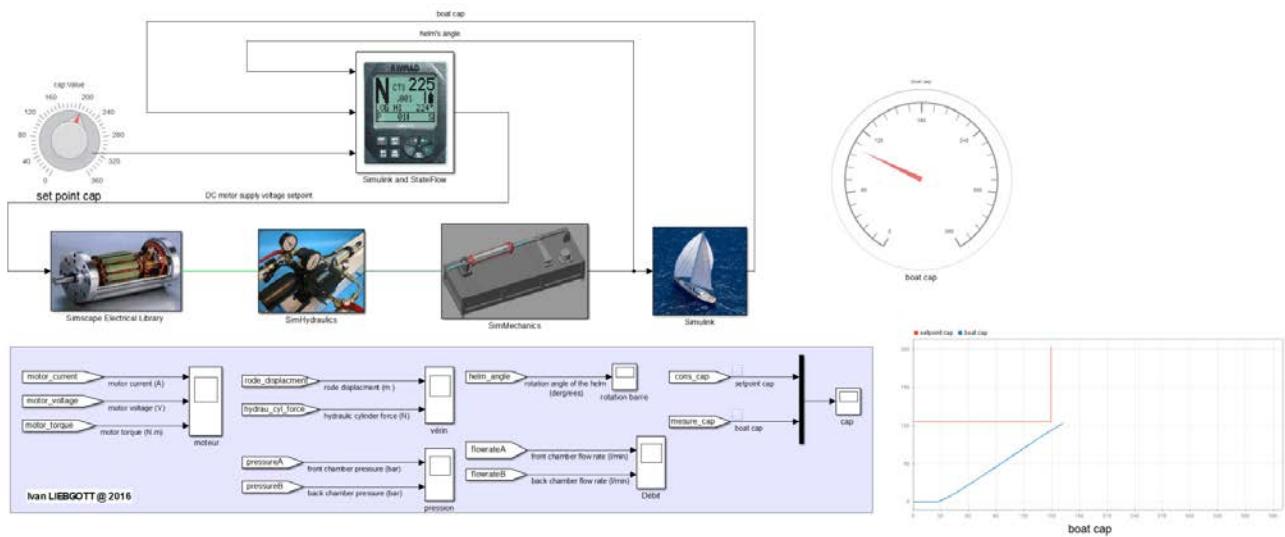


Figure 73: using interactive piloting on a model

V. Examples of multi-physical models and possible uses

A. The Maxpid robot

The Maxpid robot is a subsystem of a robot initially used to gather fruit and sort waste. Maxpid is servo controlled and can angularly orient its arm in accordance with an instruction. A spindle-nut type movement transformation system is used to change the rotational motion of the output axis of the motor (screw) in the arm rotation movement.

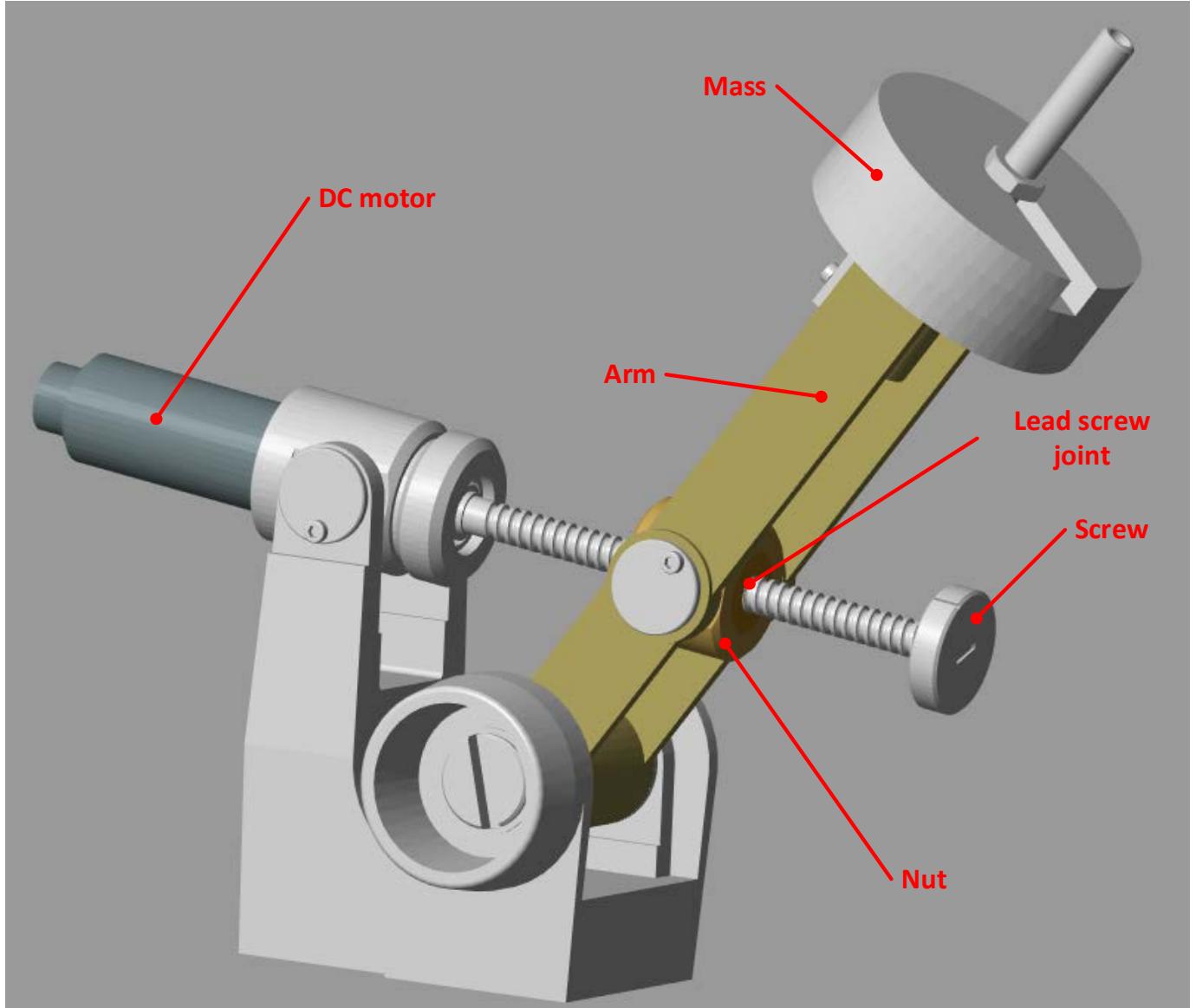


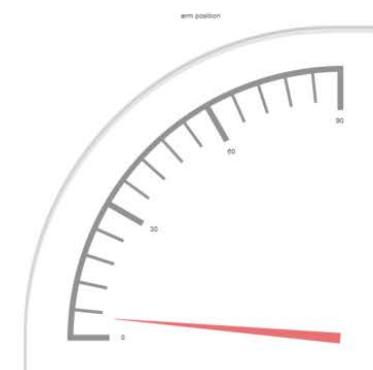
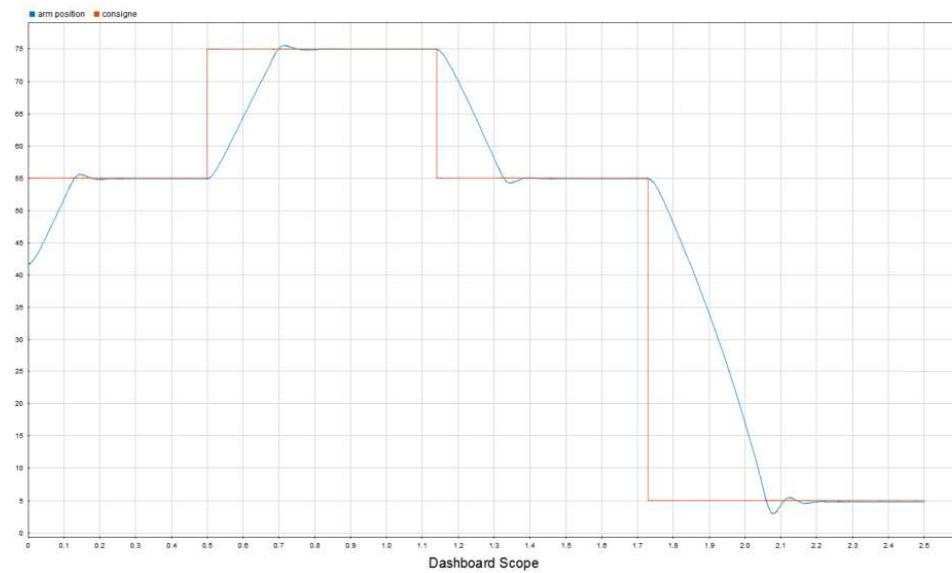
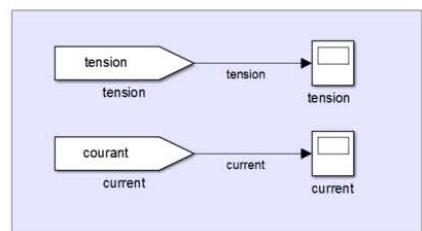
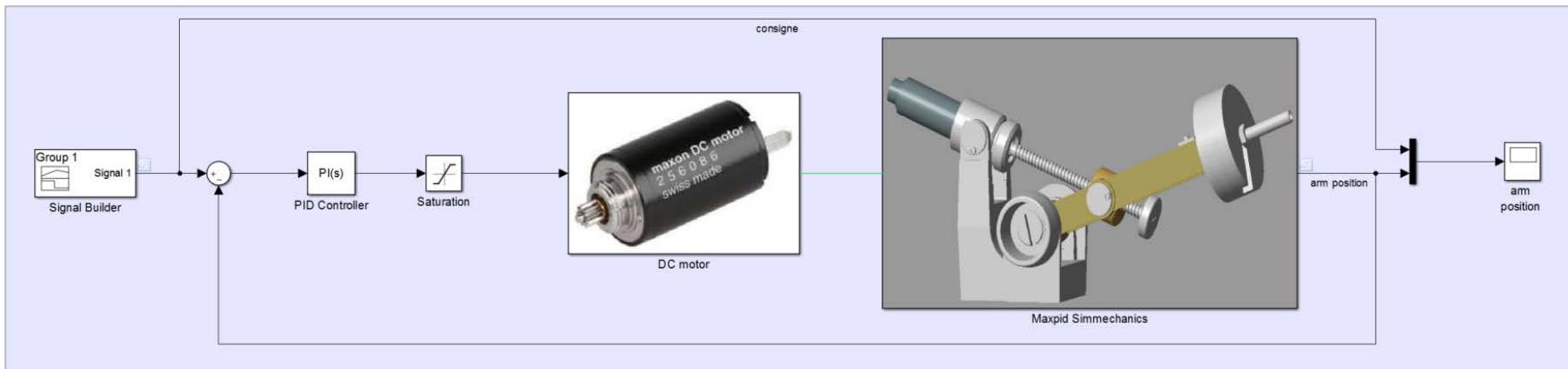
Figure 74: presentation of the Maxpid robot

The need to study the Maxpid arm position servo control requires a validated model. In this case the model is very difficult to implement analytically because the mechanical part is complex and the differential equations that characterize the behavior are nonlinear. Solving these equations is possible only by considering small variations around a given position which is very restrictive to the analysis process.

In this type of case, the use of the multi-physical modeling presents a clear advantage. The addition of the 3D model will take into account all dynamic effects without any calculation being necessary. The model will be valid for the entire range of angular variations in the arm position.

Open the “**Maxpid_US.slx**” file, **Explore** the subsystems and **Launch** the simulation.

The **SimMechanics Explorer** window allows you to visualize the movements of the robot’s arm.



Real-Time Pacer
Speedup = 0.2

@ Ivan LIEBGOTT 2016

Figure 75: multi-physics model of the MAXPID robot

Visualization of the position of the arm and setpoint:

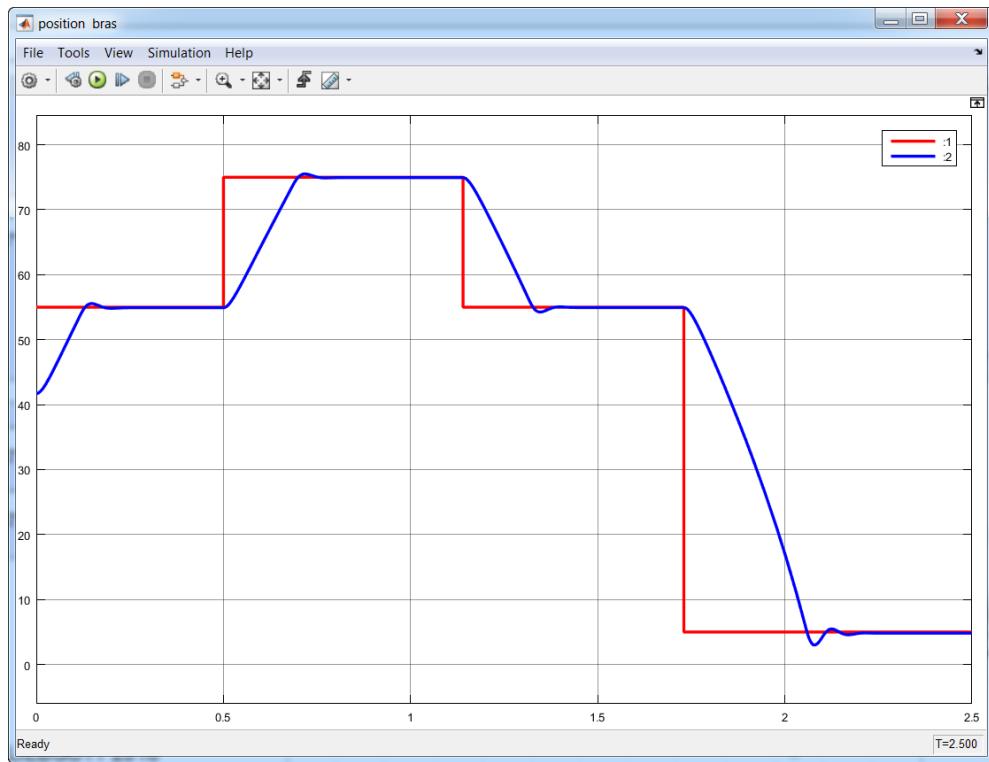


Figure 76: position of the MAXPID robot arm

Visualization of the control voltage:

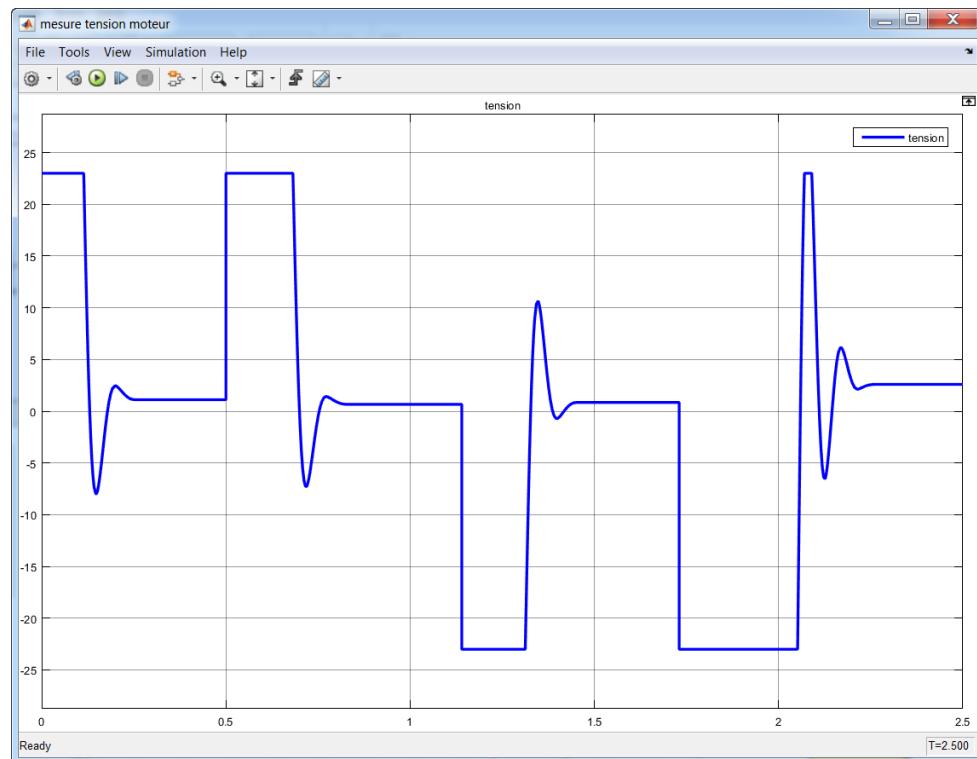


Figure 77: Visualization of the control voltage for the motor

Visualization of the current in the motor armature:

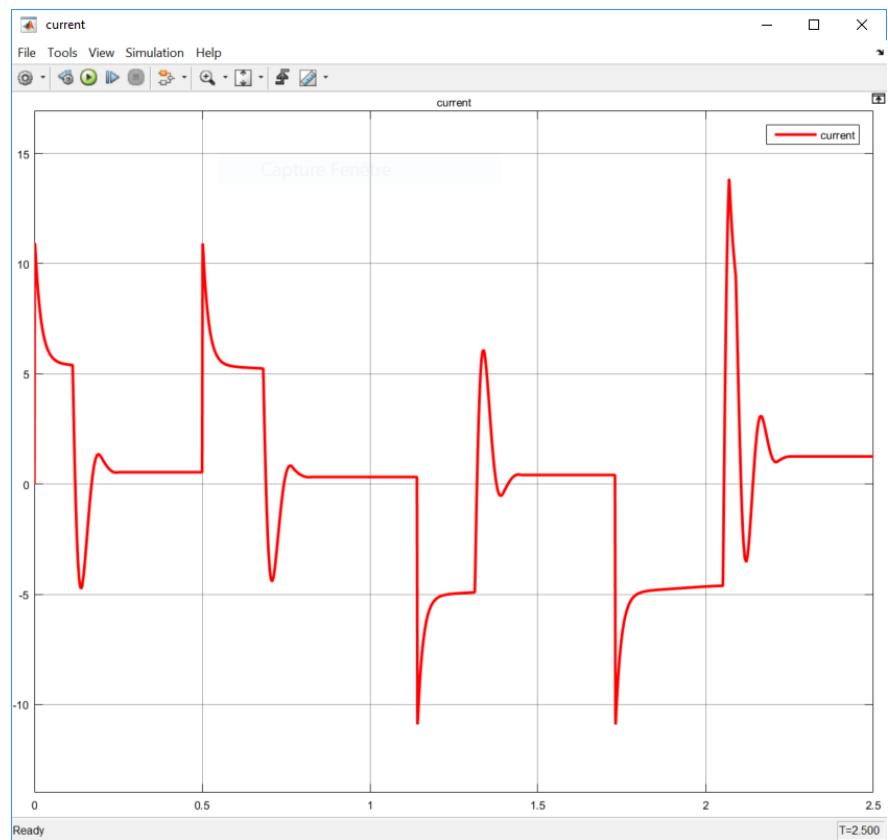


Figure 78: Visualization of the current in the motor armature

You can change the **SimMechanics** mass settings directly in order to see the effect of adding mass at the end of the arm, and act on any parameter.

B. The linear axis Control'X

The Control'X system is a servo controlled linear axis. A DC gear motor is coupled with a pulley/belt movement transformation system. The axis carriage is secured to the belt and moves in translation.



Figure 79: Control'X linear axis

Open the “**ControlX_US.slx**” file and **Explore** the subsystems.

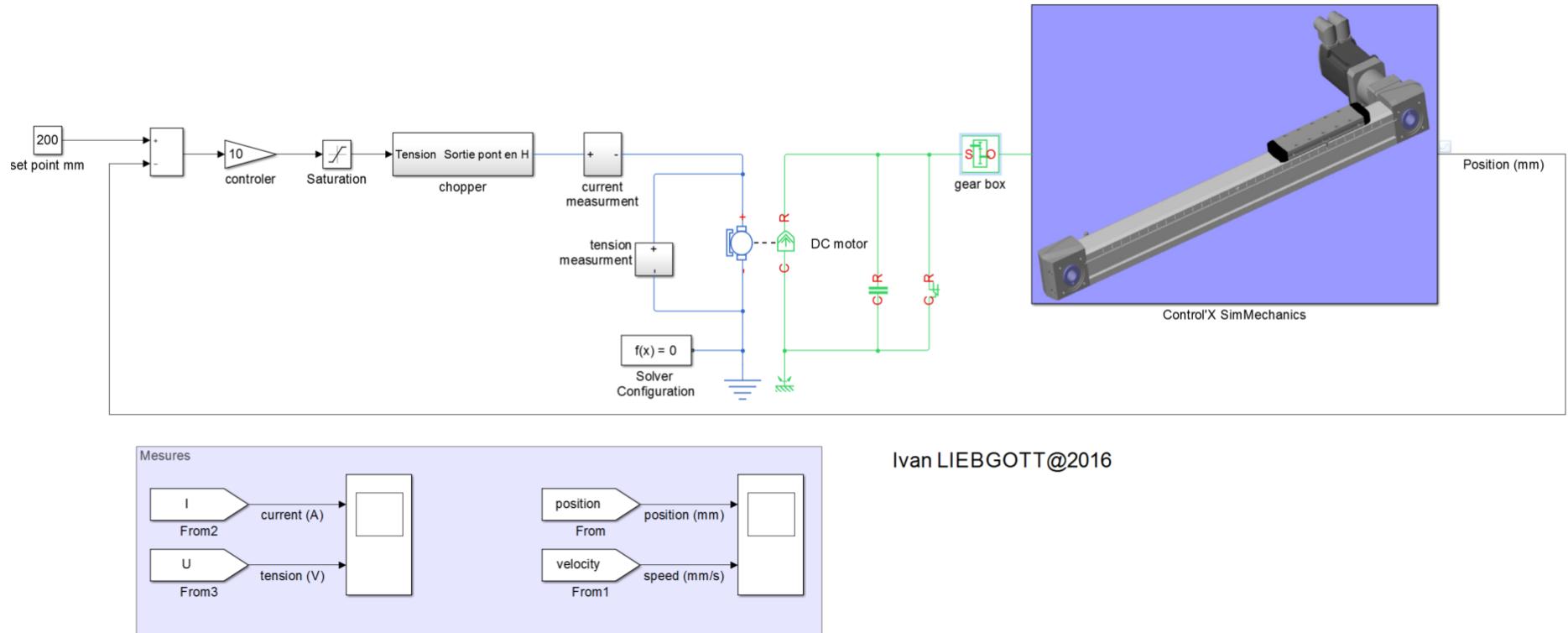


Figure 80: multi-physical model of the Control'X system

Launch the simulation and observe the results.

Visualization of the speed and the position of the axis:

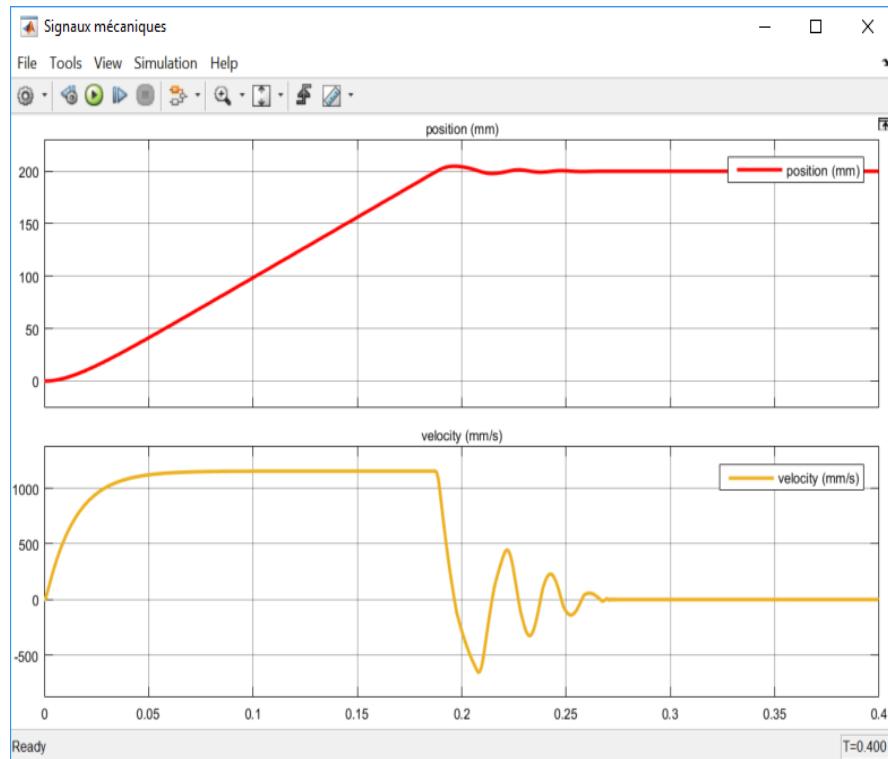


Figure 81: Visualization of the speed and the position of the linear axis

Visualization of the control voltage of the motor and the current in the armature:

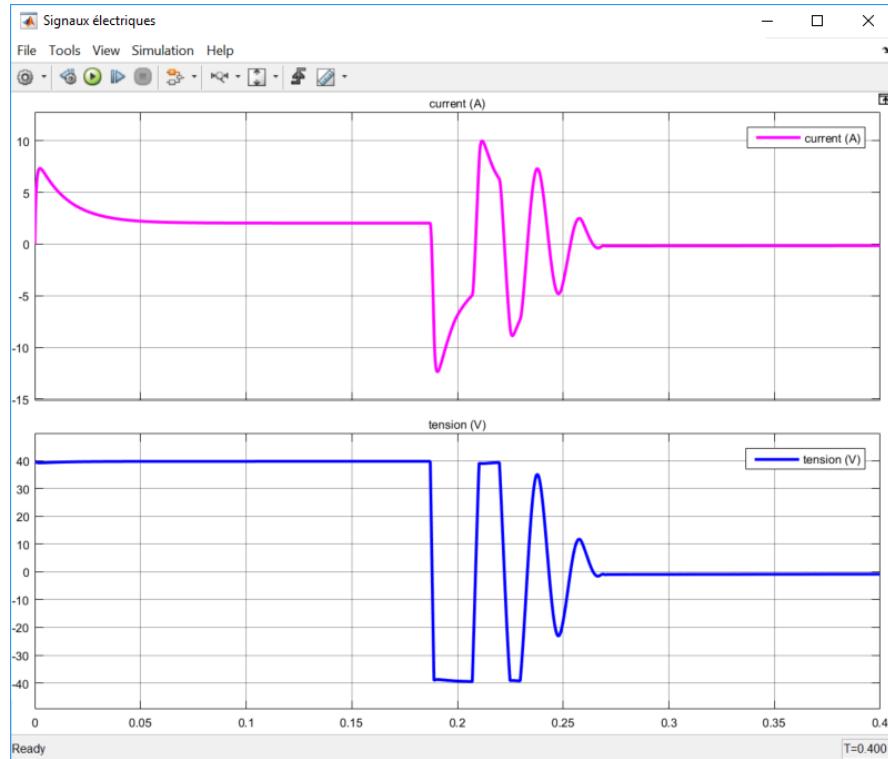


Figure 82: View of the command voltage of the motor and the current in the armature

Model validation can only be accomplished through comparison with experimental results. The “**ControlX_compar_reel_modele.slx**” file contains a comparison between the simulation performance and the performance taken from the actual system.

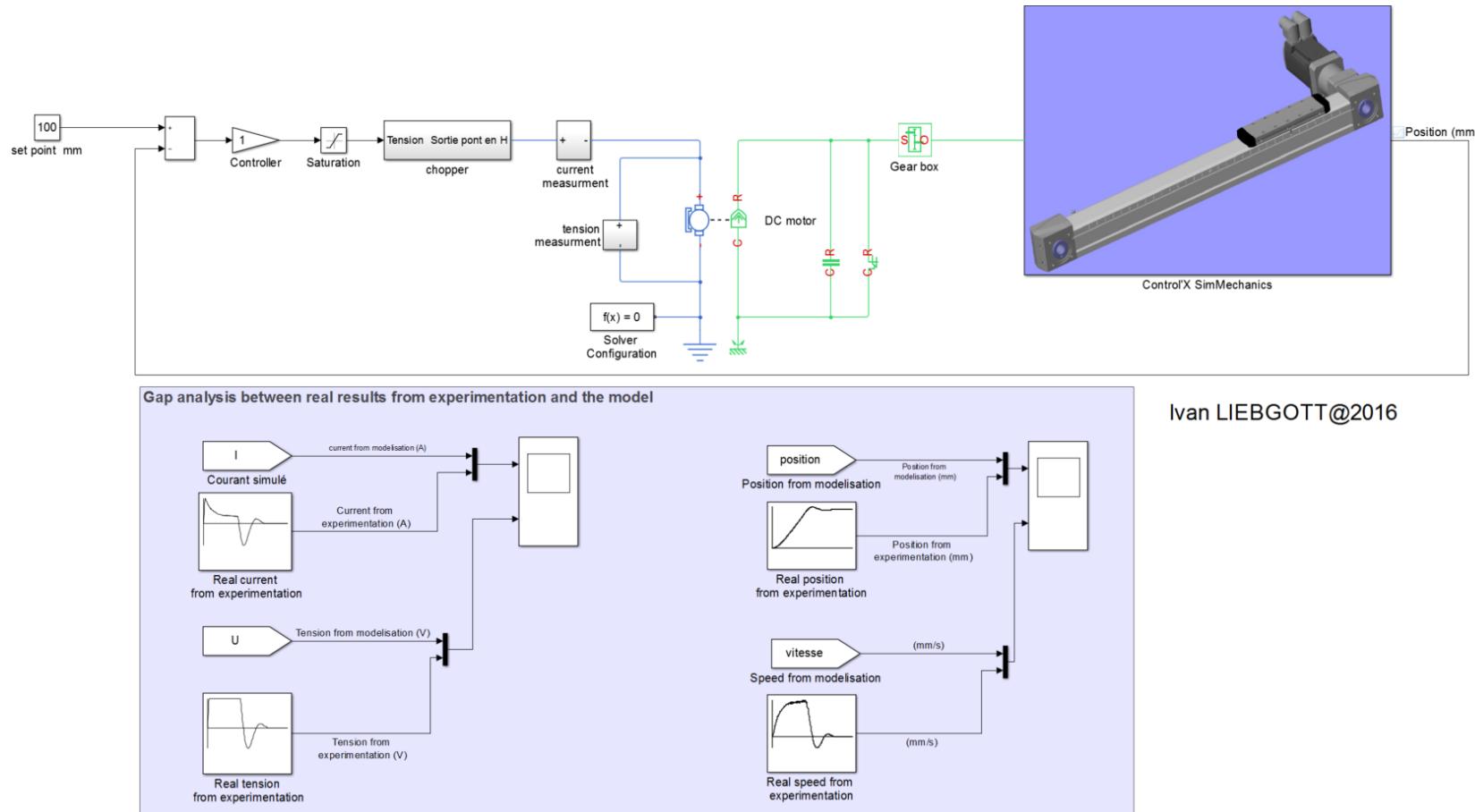


Figure 83: evaluation of the difference in performance between the model performance and those of the actual axis

Launch the simulation and observe the results.

Evaluation of the differences between model/reality for position and speed:

Figure 84: Evaluation of the differences between model/reality for speed and axis position

Evaluation of the differences between model/reality for position and speed:

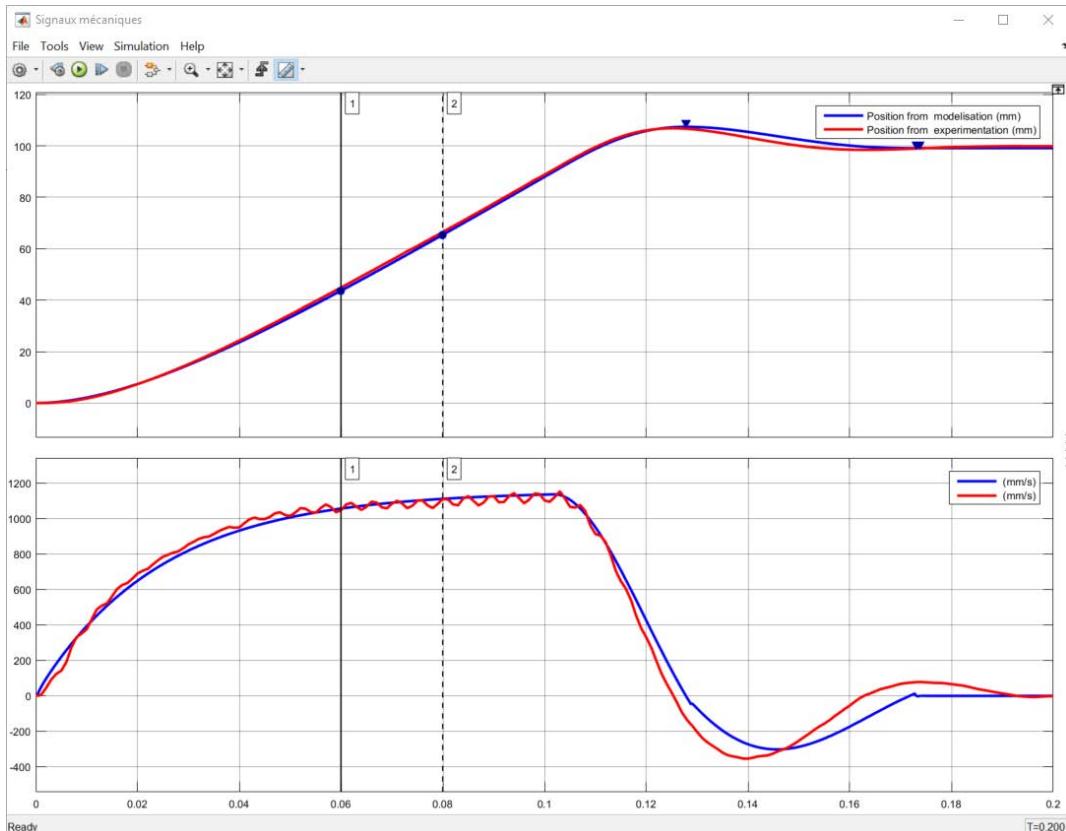


Figure 85: Evaluation of the differences between model/reality for armature current and command pressure

The differences are very low which highlights the relevance of the multi-physical modeling approach. It will be possible to build simple models which behave in a very similar way to actual systems.

C. How to make a multi-physical model with MATLAB-Simulink

The multi-physical model involves many modeling tools. To construct a multi-physical model, you need have a basic understanding of the following tools:

- MATLAB
- Simulink
- Simscape and its core libraries (mechanical, electrical, hydraulics...)
- SimMechanics
- Stateflow

You should also be able to use components of specific libraries based on modeling requirements:

- SimHydraulics
- SimPowerSystems
- SimElectronics
- SimDrivelines

This book provides a method for gradually learning these tools through examples that will be usable in the for engineer training. The first part introduces **Simscape** and modeling by components (acausal). Then, we will look at the basics of programing in **MATLAB** and how **MATLAB** and **Simulink** can communicate. Later, there is an introduction to using **Stateflow** and **SimMechanics**. The end of the book presents interesting analysis tools which are directly related to the design processes proposed in the book, such as "**System Identification Toolbox**" or "**Simulink cControl Design**" that allow for **control command** studies of servo systems. Design methods and corrective adjustment are presented using interactive graphical tools particularly suited to the formation and understanding of the phenomena.

Mastering these tools will address many of the modeling problems that may be encountered in during engineer training.

Chapter 3: Getting to grips with Simscape

This chapter aims to present the core theories in acausal multi-physical modeling using **Simscape** and its libraries.

I. Introduction to acausal modeling with Simscape

Within the **MATLAB-Simulink** environment, the **Simscape** tool offers an acausal modeling approach and has an extensive library of elementary components whose physical behavior is already modeled. The acausal modeling approach assembles components to model the behavior of a system. The calculation principle is based on a power balance in each node of the model and not based on the principle of causality, hence the name acausal modeling.

Let's take an R-L circuit as an example.

At $t = 0$ s, a voltage source feeds an RL circuit. The model should allow for the analysis of the evolution of the current $i(t)$ which flows through the circuit.

$$R=20 \Omega$$

$$L=0.01 \text{ mH}$$

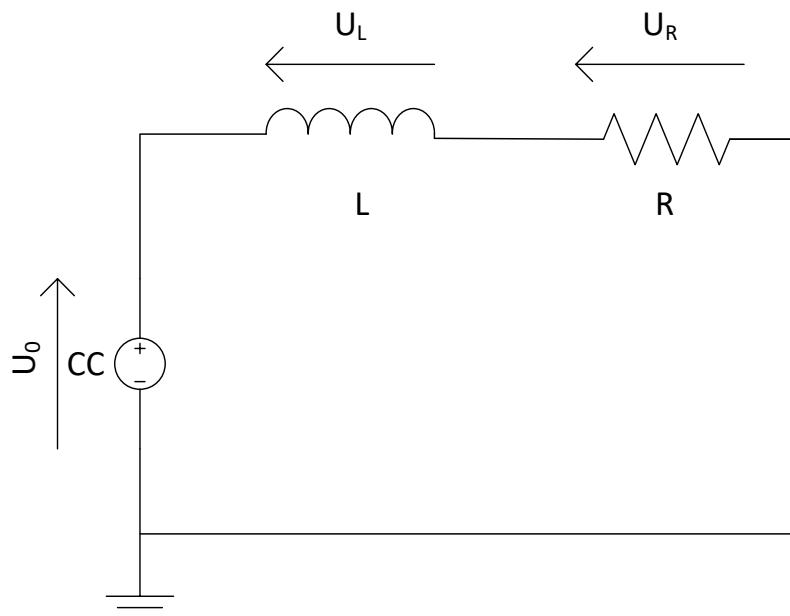


Figure 86: R-L circuit

A. Choice of components

The acausal model of an RL circuit is obtained using the following components:

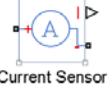
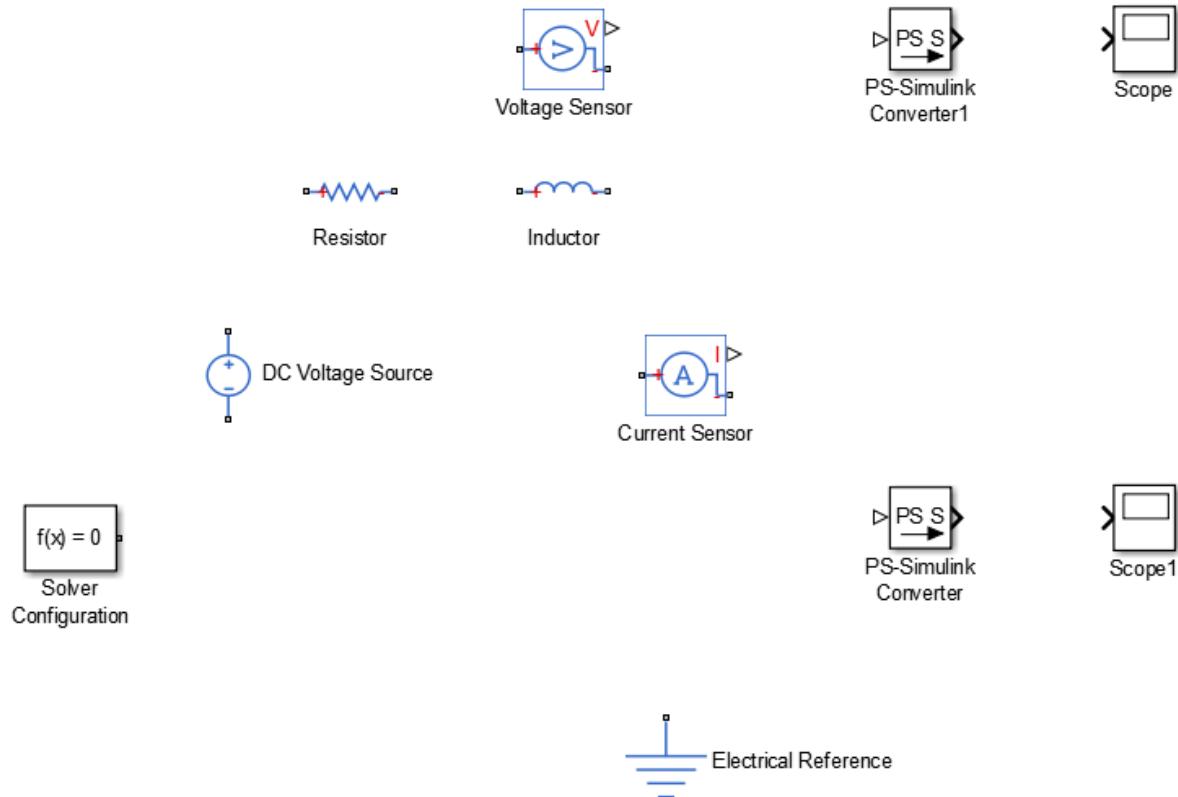
Component function	View	Library
Source of constant voltage	 DC Voltage Source	Simscape/Foundation Library/Electrical/Electrical Sources
Resistance	 Resistor	Simscape/Foundation Library/Electrical/Electrical Element
Inductance	 Inductor	Simscape/Foundation Library/Electrical/Electrical Element
Electrical reference	 Electrical Reference	Simscape/Foundation Library/Electrical/Electrical Element
Current sensor	 Current Sensor	Simscape/Foundation Library/Electrical/Electrical Sensors
Voltage sensor	 Voltage Sensor	Simscape/Foundation Library/Electrical/Electrical Sensors
Conversion of physical signal into Simulink signal	 PS-Simulink Converter	Simscape/Utilities
Monitor	 Scope	Simulink/Sinks
Solver	 Solver Configuration	Simscape/Utilities

Figure 87: components required for modeling the R-L circuit with Simscape

B. Placement and assembly of components

In the **Simulink Library Browser** window, execute the **File/New/Model** command  to create a new file.

Slide/Move the various blocks from the libraries and place them in the working window, as shown below. You can use Figure 88 to gain an understanding of which commands are useful for directing components.



Useful commands

Functions	Actions	Keyboard shortcuts
Rotating a component clockwise	Right click on the component, Rotate&Flip/Clockwise	Ctrl+R
Rotating a component counterclockwise	Right click on the component, Rotate&Flip/CounterClockwise	Shift+Ctrl+R
Left/right flip of a component	Right click on the component, Rotate&Flip/Flip Block/Left Right	Ctrl+I
Up/Down flip of a component	Right click on the component, Rotate&Flip/Flip Block/Up Down	
Copy a component	Right-click on the component you are going to copy, then slide/place keeping the right button clicked	
Delete a component	Right-click on the component, then select Delete	Delete

Figure 88: useful commands

Connect the components to create the model below:

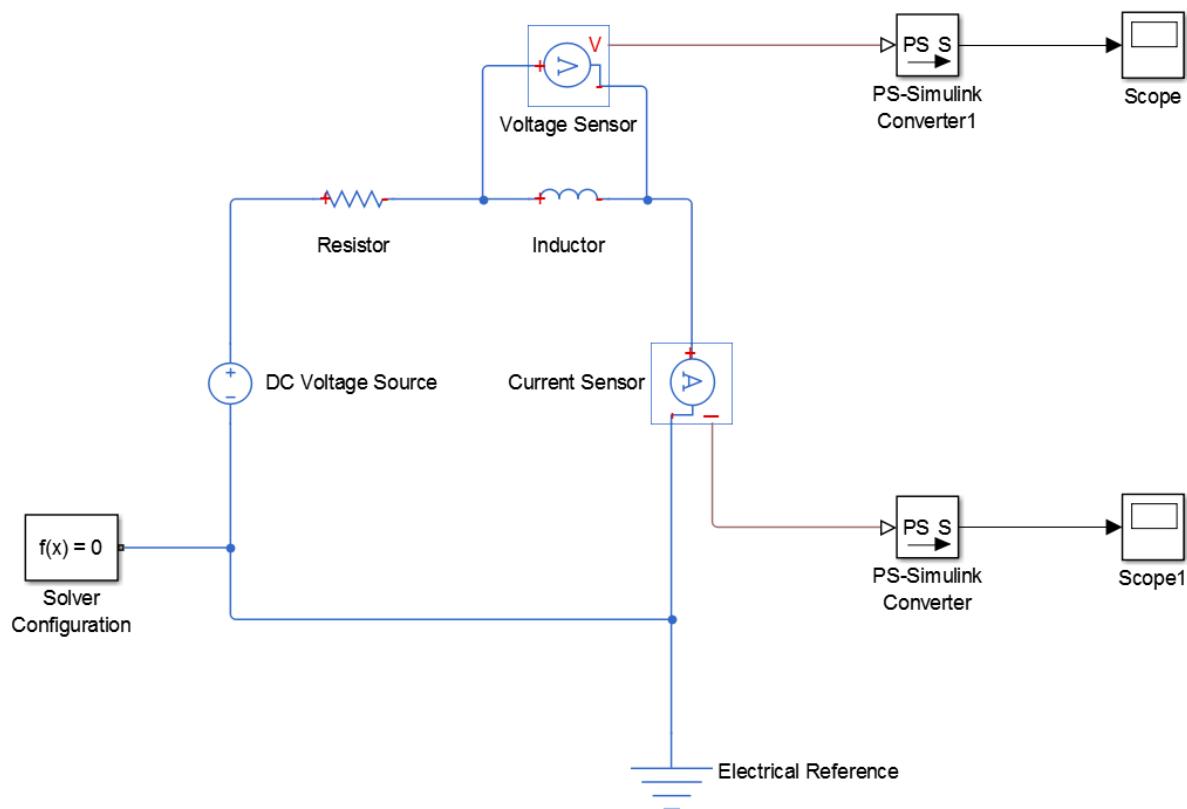


Figure 89: Simscape Model of R-L circuit

Useful commands		
Functions	Actions	Keyboard shortcuts
Connect two components	Left-click on the port of the first component and then move it, keeping the left button clicked, to the port of the second component.	

Figure 90: useful commands

C. Different types of ports and connections

Before applying the settings to the blocks and launching the simulation, let's look at the different connections between the blocks in the model. The current model uses different types of ports and connections. Ports can be placed in three categories (Figure 91).

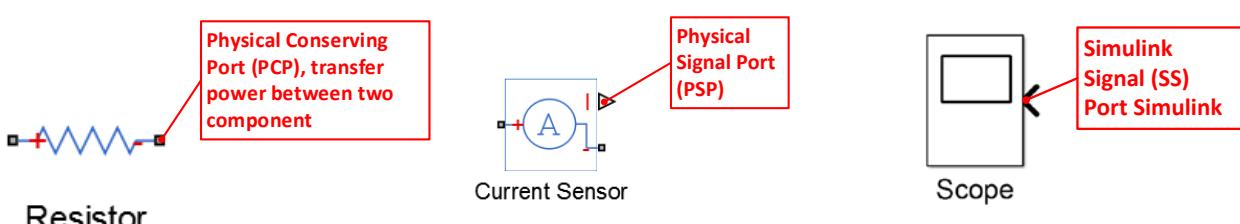


Figure 91: different port types in Simscape

- “Physical Conserving Ports” (PCP) transfer power between two components of the same domain. The connections on these ports are not oriented and are analogous in relation to the links that connect the components in reality and are in the field of acausal modeling. In the case of our model, which involves only the electric field, these connections are electrical wires. They are traversed by a current and it is possible to measure a potential difference between two points belonging to these connections. Measuring the current in the circuit is done by a current sensor placed in series in the circuit. The voltage at the coil terminals is measured by a voltage sensor positioned in parallel to the coil terminals.
- “Physical Signal Ports” (PSP) transmit physical signals collected by sensors on the model. These signals are oriented and are the image of the physical parameters taken. These ports and connections function according to the principle of causality.
- “Simulink Signal” (SS) ports transmit digital oriented signals which are readable by the Simulink library blocks. These ports and connections function according to the principle of causality.

The type of connection can be identified by the port with which it is associated.

Model building requires a perfect understanding of the nature of the information and signals which are transmitted through the different connections, as well as identifying the nature of the block ports involved in the modeling. Different types of port cannot be connected. Furthermore, two **Physical Conserving Ports** cannot be joined unless they are of the same physical field. In order to bring more clarity to models made with Simscape each physical field is represented with a different color.

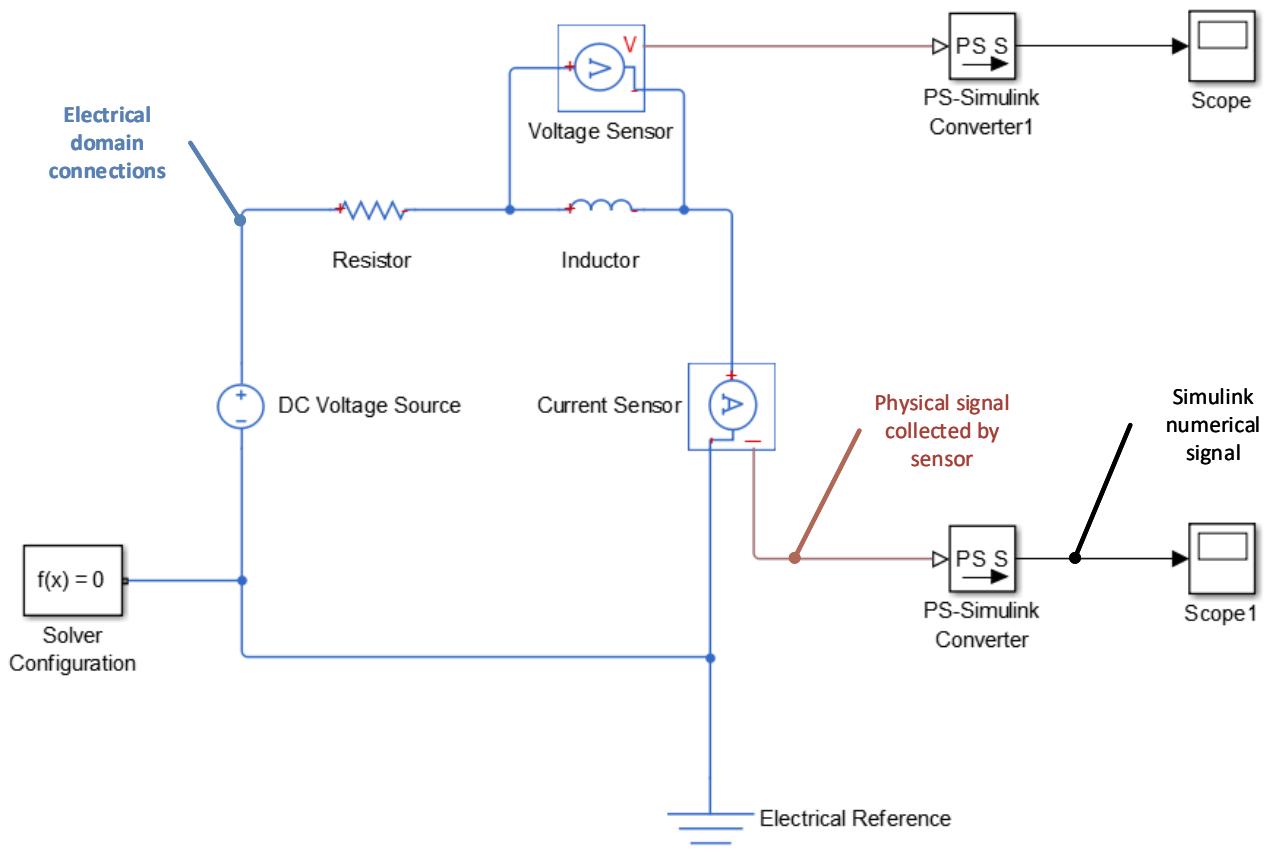
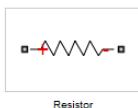


Figure 92: identification of the connections in Simscape model

D. Configuration of components

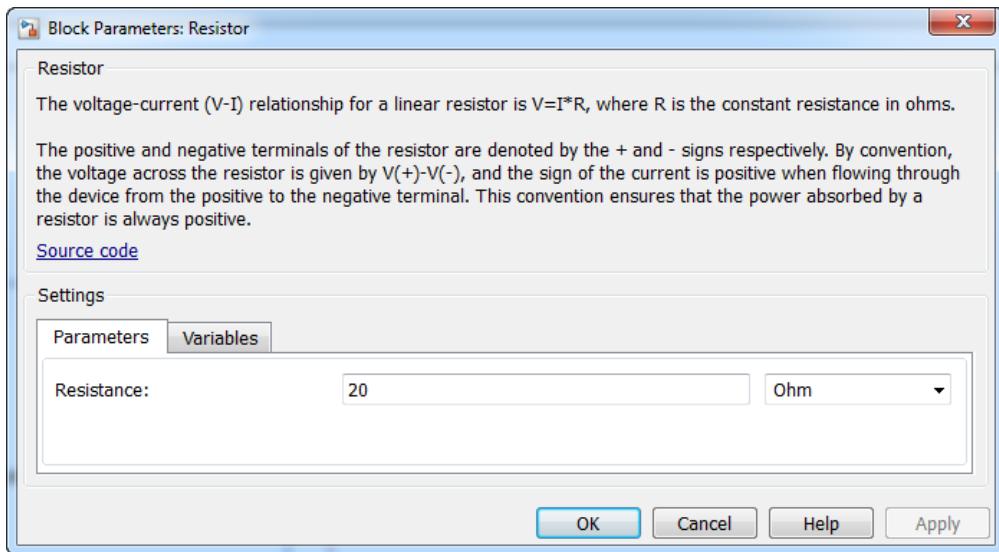
Component configuration is accessible by double clicking on the block.

RESISTOR

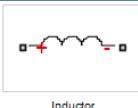


Simscape/Foundation Library/Electrical/Electrical Element

This component is configured by entering the resistance value

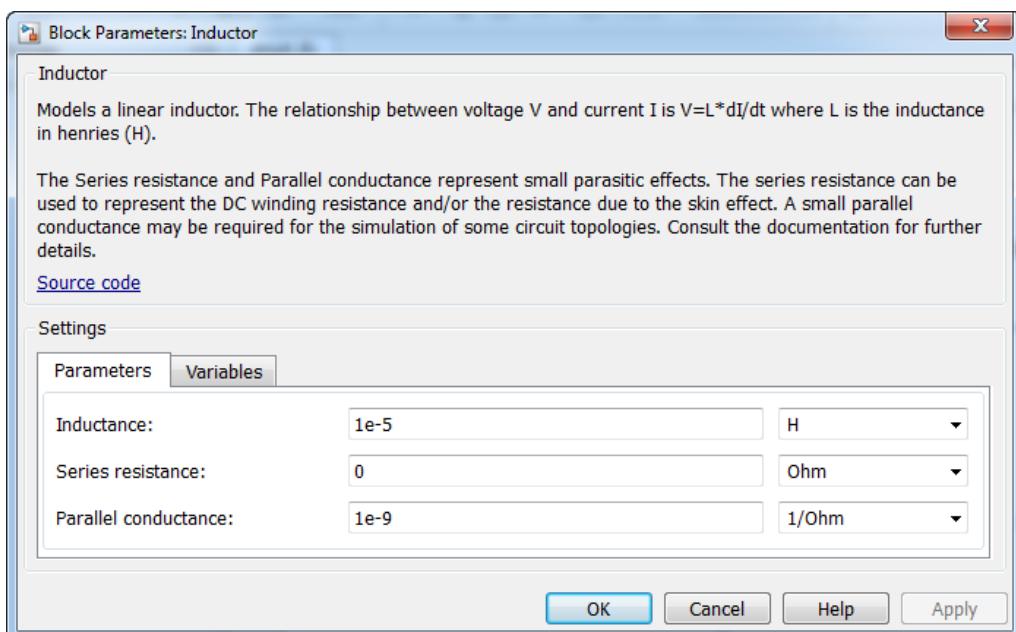


INDUCTOR



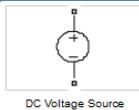
Simscape/Foundation Library/Electrical/Electrical Element

This component is configured by entering the inductance value



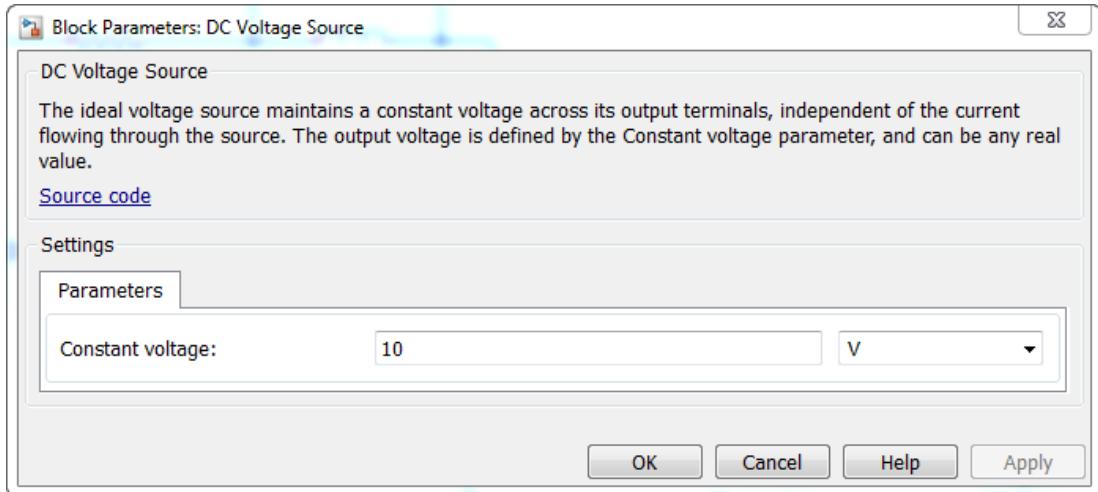
Configuration

DC SOURCE VOLTAGE



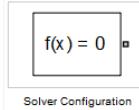
Simscape/Foundation Library/Electrical/Electrical Sources

This component is configured by entering the supply voltage value



Configuration

SOLVER CONFIGURATION

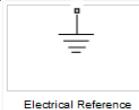


Simscape/Utilities

No modification is applied to the configuration of this block for this model. A **Solver Configuration** block must be connected to the model so that the solver can perform calculations.

Configuration

ELECTRICAL REFERENCE

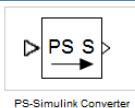


Simscape/Foundation Library/Electrical/Electrical Element

No configuration is necessary for this component. A physical field reference block also needs to be present in the model.

Configuration

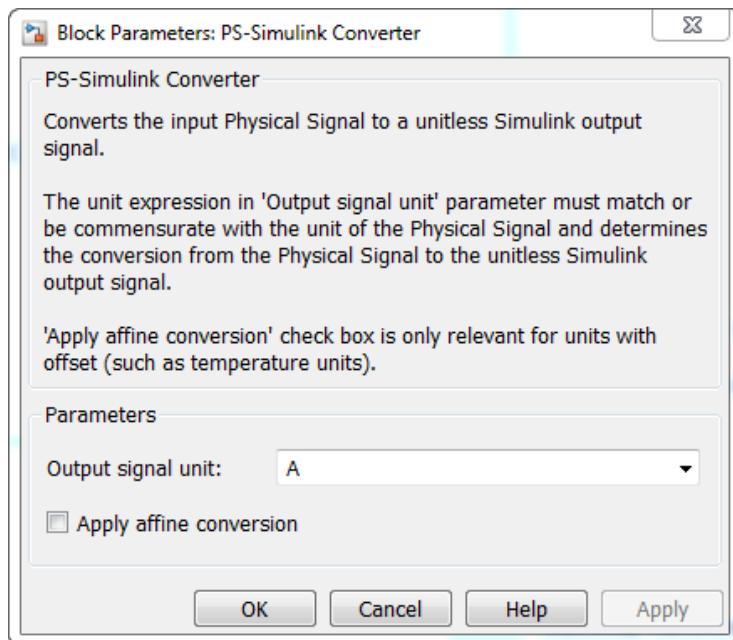
PS-SIMULINK CONVERTER



Simscape/Utilities

This component allows the conversion of a physical image of the parameters measured by the **Simulink** signal so that it can be displayed in a scope. It is desirable to specify the block of the prefeed physical variable for display in Simulink. Indicate in the two corresponding blocks that we want to measure the current in amperes (A) and voltage in volts (V). The dropdown menu offers a choice of blocks, we can see that it is possible to expand the choice of blocks. If the block is not specified by the user, the SI block for the field will be selected by default.

The functionality of this block is useful in mastering the identified blocks of physical parameters. Simscape automatically manages the consistency of blocks for the resolution of the model and the user does not have concern themselves with this.



E. Launching the simulation and analyzing the results



If necessary, the complete model is available on the in the file **circuit_RL.slx**

Specify a simulation time of between **1 and 5 seconds** and **launch** the simulation by clicking on



Double-click on the scope which measures the current to obtain the variation over time.

Click on the automatic scaling setting if necessary to obtain the following curve:

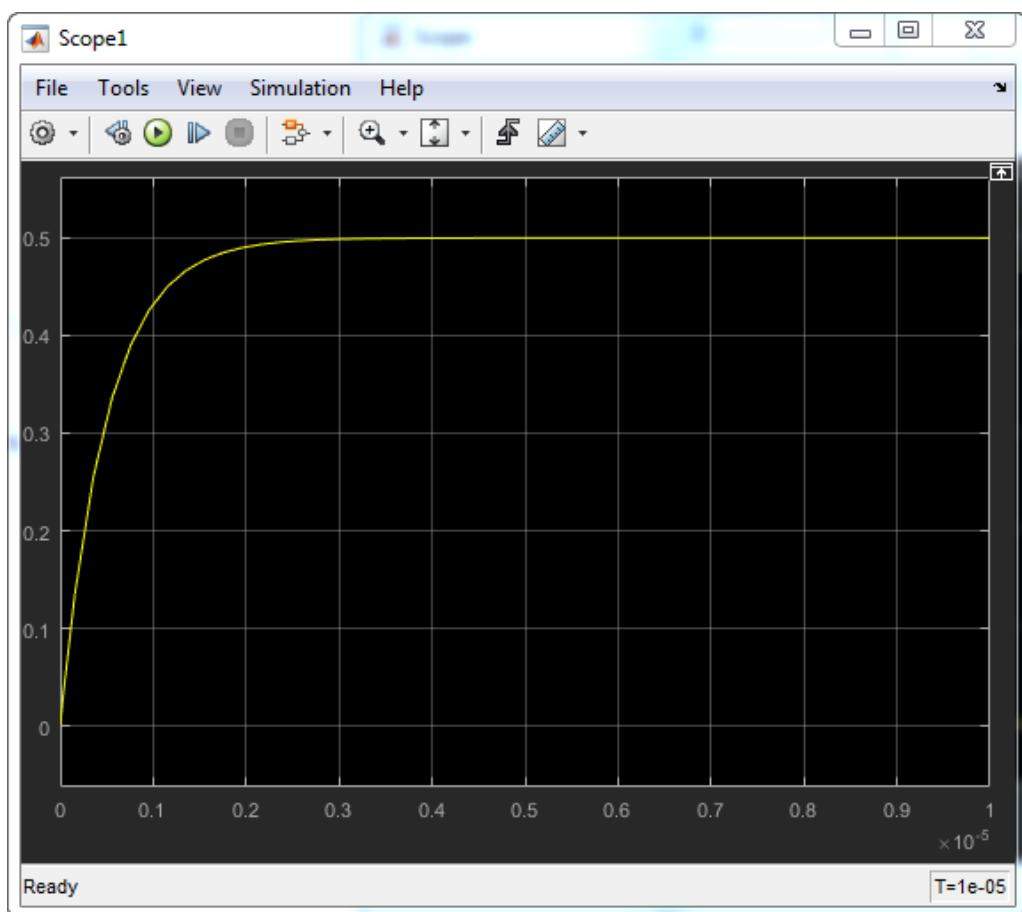


Figure 93: evolution of the intensity of current in an RL circuit

Refer to appendix "Setting Curves" to shape the curves.

The procedure is identical to visualize the temporal evolution of the voltage at the coil terminals.

Click on the automatic scaling setting  if necessary to obtain the following curve:

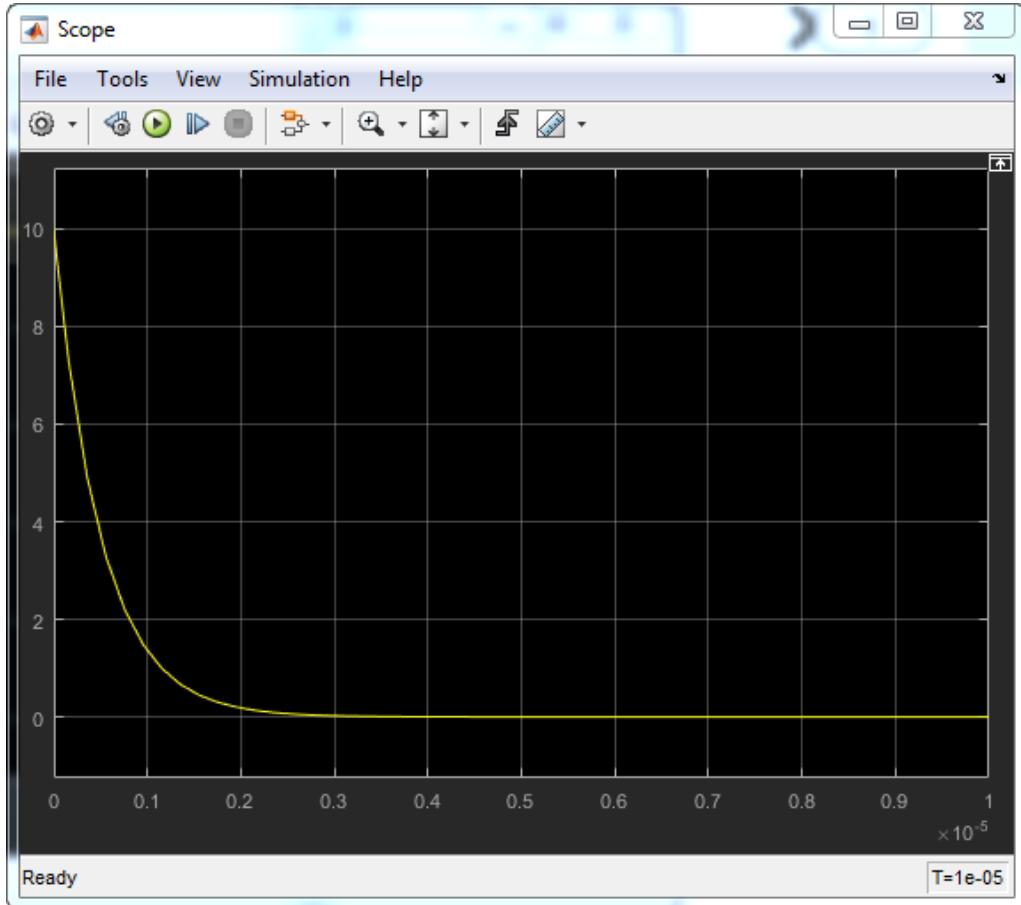


Figure 94: evolution of the voltage at the coil terminals in an RL circuit

You can change the inductance value of the coil to see its influence on the establishment of the current in the circuit by changing the **inductor** setting block and restarting the simulation.

We can see that this modeling approach is very intuitive and allows you to build and modify a model very quickly. The model has a build structure very close to that which can be observed on the real system, which gives excellent readability.

II. Comparison with the causal approach

A. Equation for system behavior

The voltages in the circuit are related by the following temporal relationship:

$$U(t) = U_R(t) + U_L(t) = R i(t) + L \frac{d i(t)}{dt}$$

Applying the Laplace transform to this equation and assuming the Heaviside conditions:

$$U(p) = (R + Lp) I(p)$$
$$I(p) = \frac{1}{(R + Lp)} U(p)$$

The evolution of the current is given by the time response to a voltage level of a first order transfer function.

B. Choice of components

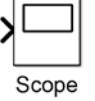
Component function	View	Library
Source level	 Step	Simulink/Sources
Transfer function	 Transfer Fcn	Simulink/Continuous
Monitor	 Scope	Simulink/Sinks

Figure 95: the components required for modeling the R-L circuit with Simscape

C. Placement and assembly of components

In the **Simulink Library Browser** window, execute the **File/New/Model** command to create a new file.

Slide/Move the various blocks from the libraries and place them in the working window, as shown below.

Connect two components. You can use Figure 97 to gain an understanding of which commands are useful for connecting and ordering **Simulink** blocks.

The resulting model is as follows:

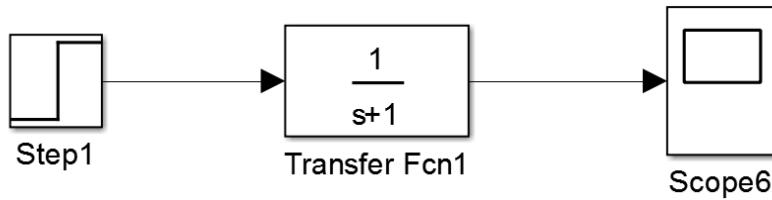


Figure 96: R-L circuit model created with Simulink

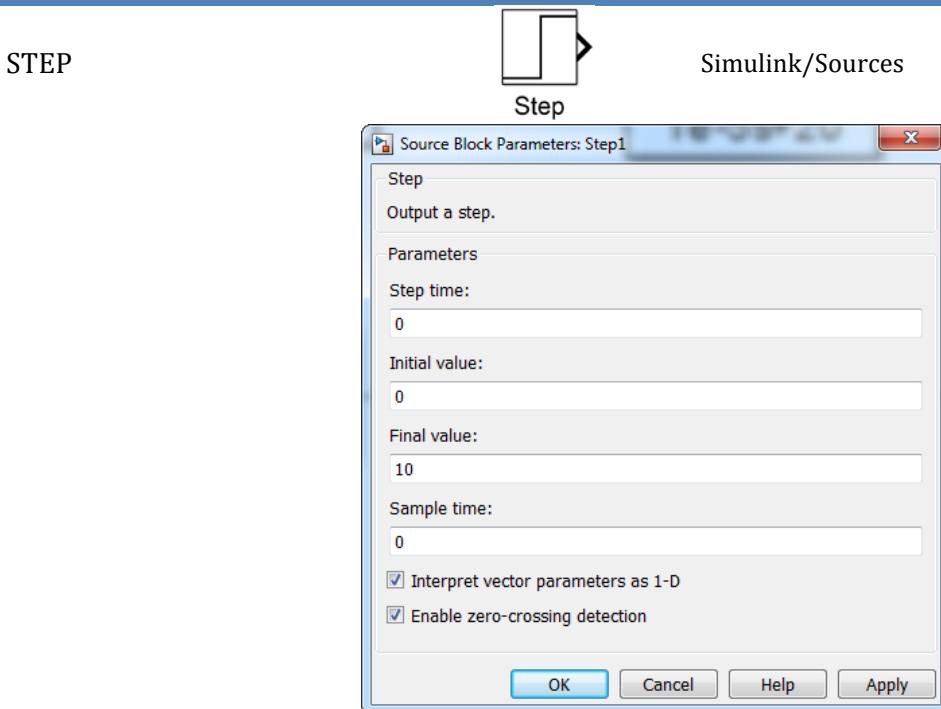
Useful commands

Functions	Actions
Connect two components	Select the first component. Select the second block whilst pressing the ctrl button. The blocks are automatically connected.
Align the components	Select the components concerned. Then right-click, Arrange/Align Middle
Order the position of the components	Select the components concerned. Then right-click, Arrange , then choose the desired order

Figure 97: Useful commands

D. Configuration of components

Configuration



Step time: moment at which the level passes the initial value, here at t=0s.

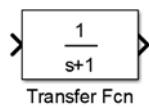
Initial value: initial value of the level, here 0

Final value: final value of the level which corresponds here to a voltage of 10V, which is to be used.

Sample time: always leave this value at 0 for the study of continuous time systems.

Configuration

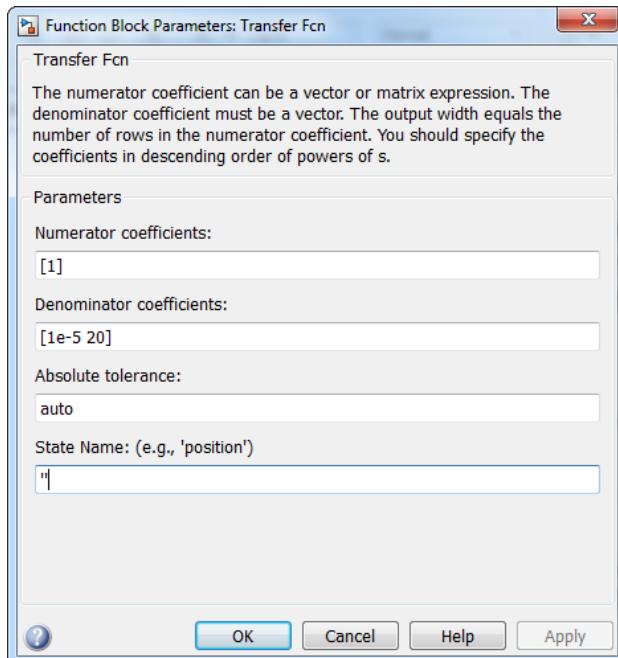
TRANSFER FUNCTION



Simulink/Continuous

$$H(p) = \frac{1}{R + Lp} = \frac{1}{1e^{-5}p + 20}$$

The transfer function to enter is



Transfer functions are configured by entering the coefficients of the numerator and denominator, starting with the monomial coefficient of the highest degree.

Example:

$$G(p) = \frac{3p + 1}{12p^2 + 5p + 4}$$

Numerator coefficients: [3 1]

Denominator coefficients: [12 5 4]

E. Launching the simulation and analyzing the results



Specify a simulation time between **1 and 5 seconds** and launch the simulation by clicking on

Double click on the scope which measures the current to obtain the variation over time.

Click on the automatic scaling setting to obtain the curve:

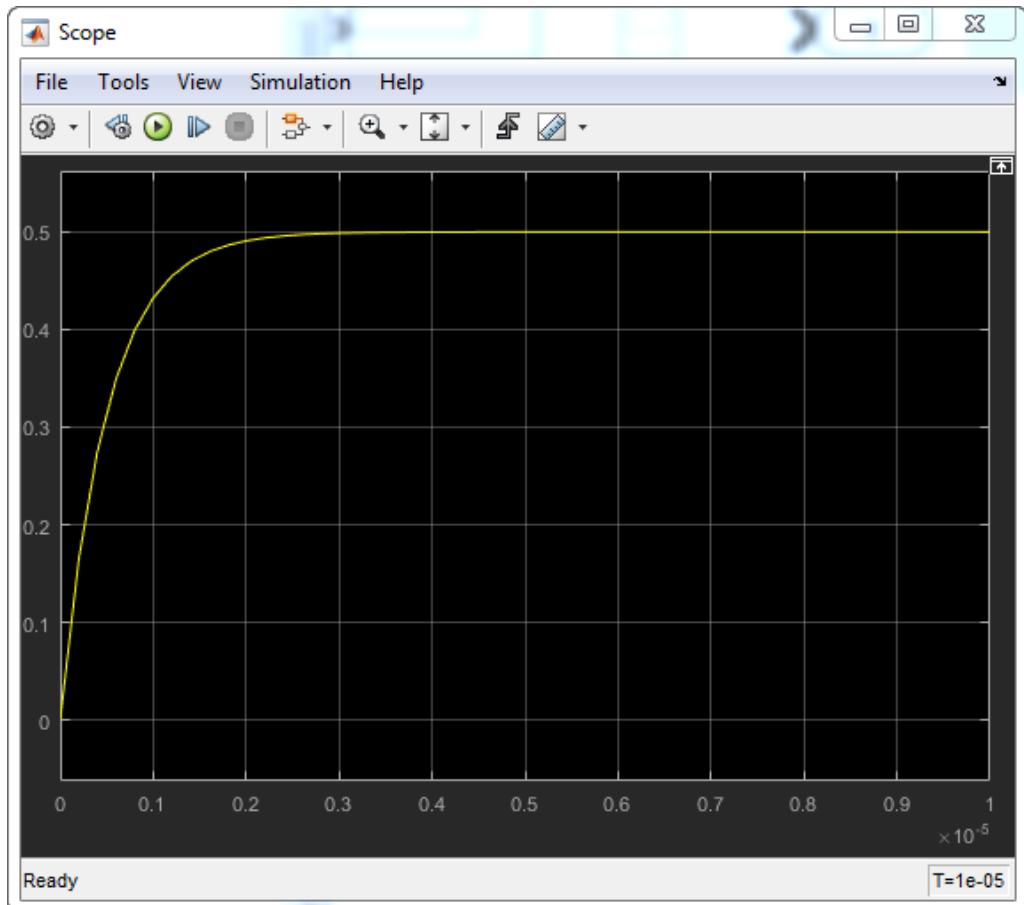


Figure 98: evolution of the intensity of current in an RL circuit

The curve showing the evolution of the current in the circuit is the same as the acausal model (thankfully!). We can see that the evolution of the voltage at the coil terminals is not directly accessible, modifying the model would however provide access.

F. Advantages and disadvantages of the causal and acausal approaches

	Acausal modeling	Causal modeling
Advantages	<ul style="list-style-type: none"> • Very close to the actual system • Easy to take into account complex physical phenomena • Easy to modify, easy to read • Quicker than other modeling methods • Initial conditions can be chosen, no matter what • All the physical parameters are accessible and measurable within the model • No need to write equations • The connections between components convey a richer level of information • No math tools necessary • Immediate recognition of the field • Digital resolution method more adapted to complex model resolution • Superficial knowledge concerning the level of modeling of the components used⁽¹⁾ • The models obtained are highly nonlinear which makes the command control process delicate⁽²⁾ • Requires basic knowledge in the physical fields concerned 	<ul style="list-style-type: none"> • Perfect control at modeling level • Model structure more suited to the establishment of a command control approach • Knowledge of introduction of non-linearity
Disadvantages		<ul style="list-style-type: none"> • The physical parameters are considered nil at t=0 (Heaviside conditions) • Difficult to read and modify • Requires a perfect theoretical understanding of the phenomena studied • Requires the use of advanced math (Laplace transform) • Modeling complex systems leads to many dead-ends in the model which poses problems in digital resolution solving.

Figure 99: advantages and disadvantages of the causal and acausal approaches

(1) The level of modeling of the component is established by the software. The understanding and interpretation of the equations used requires a good knowledge of the field to enable the component to be configured correctly. It is important to note that **Simscape** and its libraries offer various levels of modeling for the same type of component.

(2) The modeling components by default includes many nonlinearities that reflect the actual behavior of the component (threshold, saturation, hysteresis ...).

These two modeling approaches co-exist in the construction of a multi-physical model. The choice of one or the other approach will depend on the use that is desired for the model and the level of knowledge of the physical field covered.

III. Modeling fundamentals with Simscape

A. Ideas of physical fields

Various physical or technological fields co-exist within the **Simscape** environment. It is very important to analyze and identify the nature of energy flows that pass through the ports of a component and identify the physical field(s) involved in the modeling.

The number of Physical Conserving Ports (PCP) which have a component is determined by the amount of energy flow that it exchanges with the exterior. It can exchange energy flows in the same physical field or provide a conversion of power from one physical field to another.

Power conversion can be done with a yield of 100% or can take into account parameters which characterize the losses. This depends on the level of modeling of the component and the phenomena considered. In the table in Figure 100 we can see some examples of components and the associated ports.

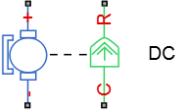
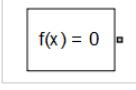
Component function	View	Types of ports
Resistance	 Resistor	This component has 2 PCP type ports from the electrical field. <u>Parameter:</u> resistance
Reducer	 Gear Box	This component has 2 PCP type ports in the mechanical rotation field. <u>Parameter:</u> reduction ratio
Mechanical rotation conversion/translation	 Wheel and Axle	This component has 2 PCP type ports. Port A is from the mechanical rotation field and Port B is from the mechanical translation field. <u>Parameter:</u> wheel radius
Conversion of electrical power into mechanical power	 DC Motor	This component has 4 PCP type ports. 2 ports (+ and -) in the electrical field and 2 ports (C and R) in the mechanical rotation field. <u>Parameter:</u> k_t en N.m/A
Source of hydraulic flow	 Hydraulic Constant Flow Rate Source	This component has 2 PCP type ports from the hydraulic field. <u>Parameter:</u> flow
Conversion of mechanical power into hydraulic power	 Fixed-Displacement Pump	This component has 3 PCP type ports. 1 port (S) in the mechanical field and 2 ports (T and P) in the hydraulic field. The mechanical port (S) transmits torque which makes the pump cylinder rotate, generating a transfer of fluid from port T to port P. <u>Parameter:</u> pump cylinder (and other related parameters characterizing the efficiency of power conversion)

Figure 100: PCP ports in Simscape components

B. Important blocks in Simscape

Models designed with **Simscape** must contain certain blocks which are necessary for the functioning of the simulation.

The “Solver” block:

Designation	View	Library
Solver	 Solver Configuration	Simscape/Utilities

A Solver block must be linked to a part of the **Simscape** model. If this block is not present, an error message will be displayed when the simulation is launched.

The “Reference” blocks for the physical fields that are used:

If a physical field is used by a **Simscape** model, the model must contain a reference to the field that is being used. If a model has various physical fields, each field must have its own reference. If a reference is not present, an error message will be displayed when the simulation is launched.

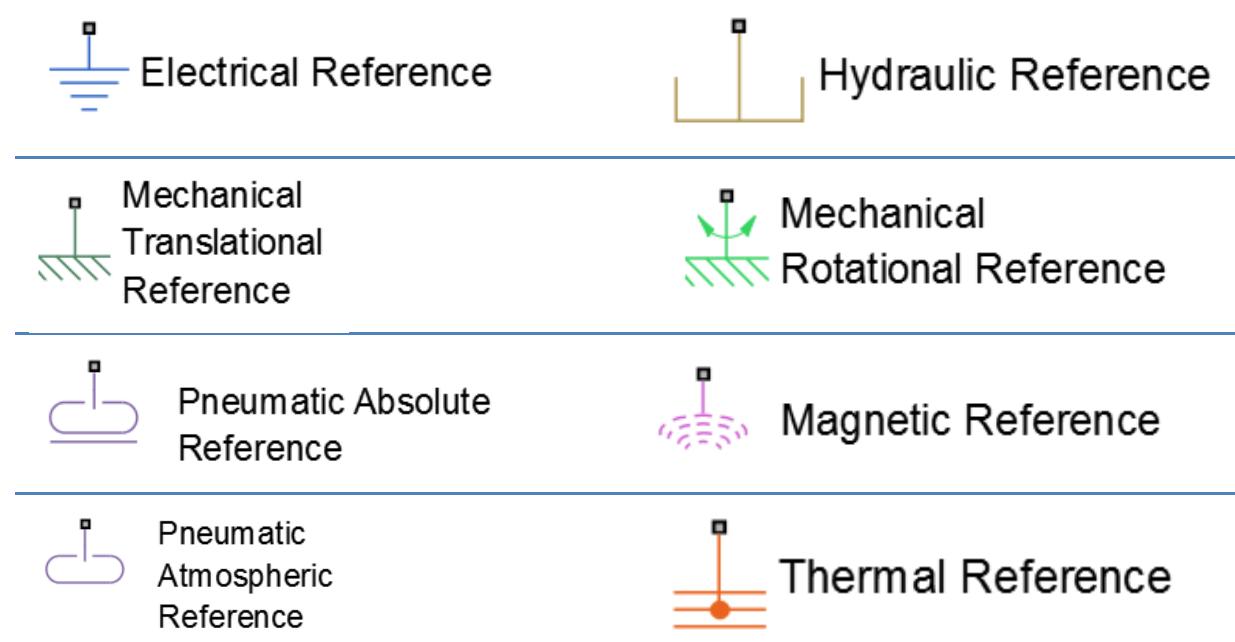


Figure 101: References for Simscape physical fields

C. “Across” and “Through” type variables and position sensors

In **Simscape**, energy flows are defined using two variables, one "**Across**" (parallel) and the other "**Through**" (series). These variables are called conjugate variables and their product characterizes the power in the relevant physical field.

- **Through** type variables are measured using a sensor placed in a **series** in the model (electrical current, hydraulic flow ...).
- **Across** type variables are measured using a sensor placed in **parallel** between two points of the model (potential difference between two points of an electrical circuit, pressure difference between two points of a hydraulic circuit ...).

It is therefore essential to know the nature (Across or Through) of the physical quantity that is to be measured on a connection on a model to the extent that this information will determine the location of the sensor. An error may cause an error message when launching the simulation or erroneous behavior of the model.

Physical field	Across Variable/s	Through Variable/s
Electrical	Voltage	Current
Hydraulic	Pressure	Flow
Mechanical translation	Linear speed	Force
Mechanical rotation	Angular speed	Time
Pneumatic	Pressure and temperature	Mass flow and entropy flux
Thermal	Temperature	Thermal flow and entropy flux

Figure 102: the Across and Through variables in Simscape

D. Orientation of components

In **Simscape**, the components have a standard orientation. The positive direction of a component is always defined from one port towards the other. This information is displayed in the configuration window for the component.

The polarity of the components needs to be taken into account in the following situations:

- The use of active components
- Placing of sensors
- Use of components for which the dynamic is orientated

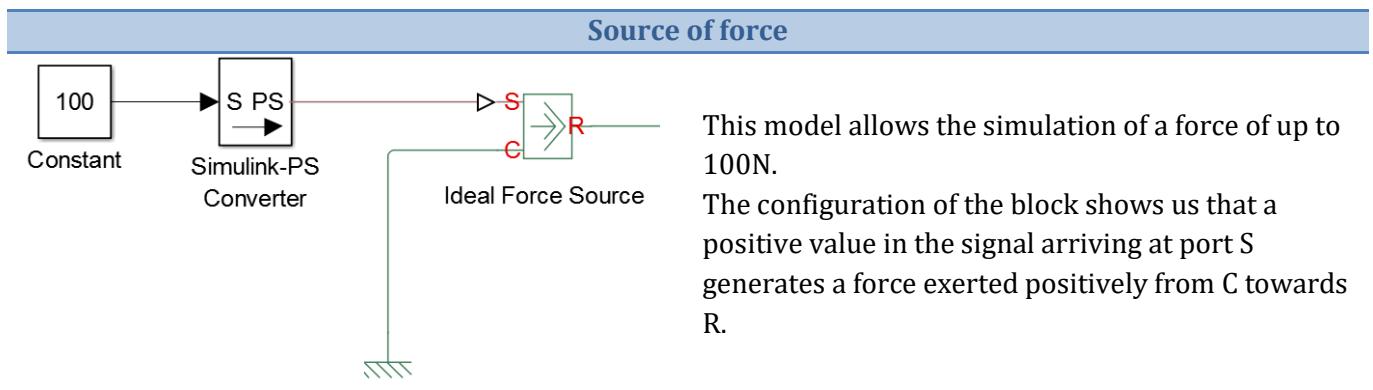
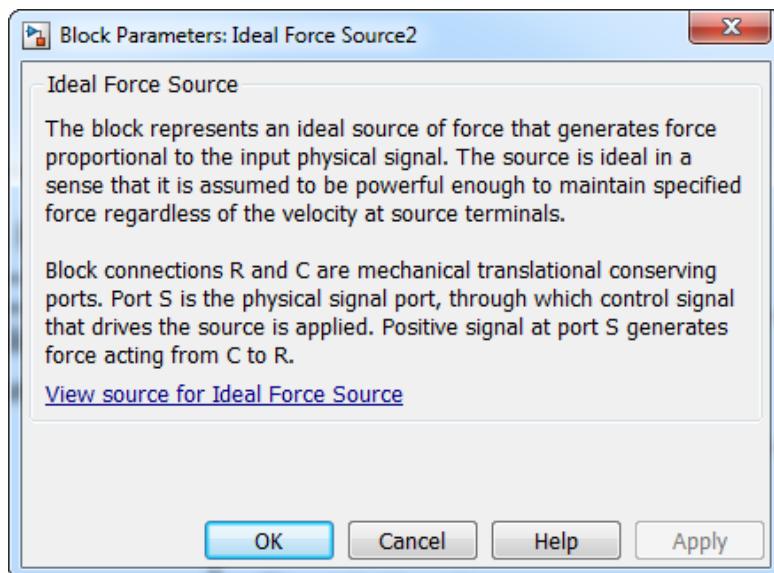
1. The use of active components

When active components are used (sources of speed, force, flow, pressure, ...), the orientation of the component must be considered.

Example of the use of a source of force

Designation	View	Library
Source of force		Simscape/Foundation Library/Mechanical/Mechanical Sources Ideal Force Source

Configuration window:



On the model in Figure 103, a force of 100N, directed from C towards R, will act on the shock damper spring.

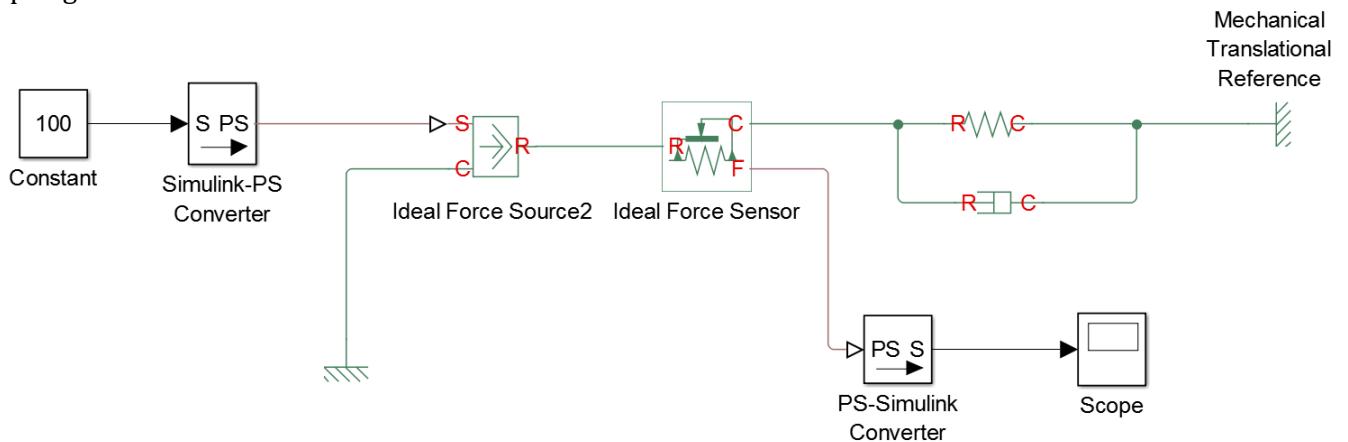


Figure 103: source of force oriented positively

On the model in Figure 104, the C and R ports have been reversed, and the force will be applied oppositely to the configuration in Figure 103.

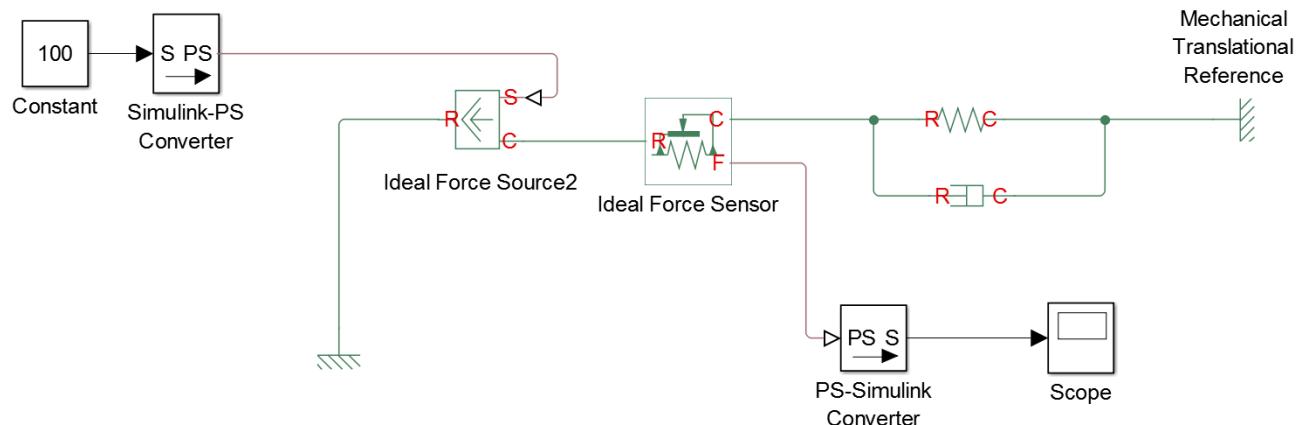


Figure 104: source of force oriented negatively

This reasoning can be generalized to relate to any type of energy source.

Open the “source_force_orientation_US.slx” file.

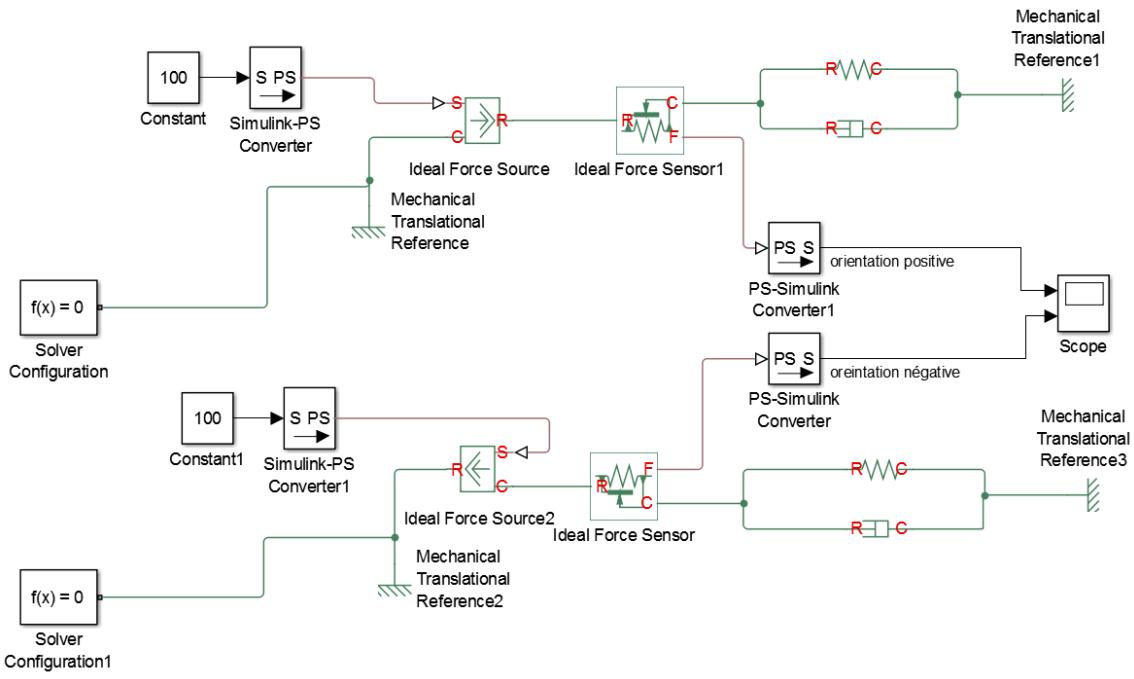


Figure 105: orientation of a source of force

This file contains two identical models of a source of force acting on a spring + damper assembly. A force sensor placed in series will measure the force (Through type variable) exerted by the source of force on the spring + shock absorber assembly. In the corresponding model under Figure 105, the orientation of the source of force has been reversed.

Launch the simulation and observe the force using the scope for the two models.

Click on the automatic scaling setting for the scope if necessary

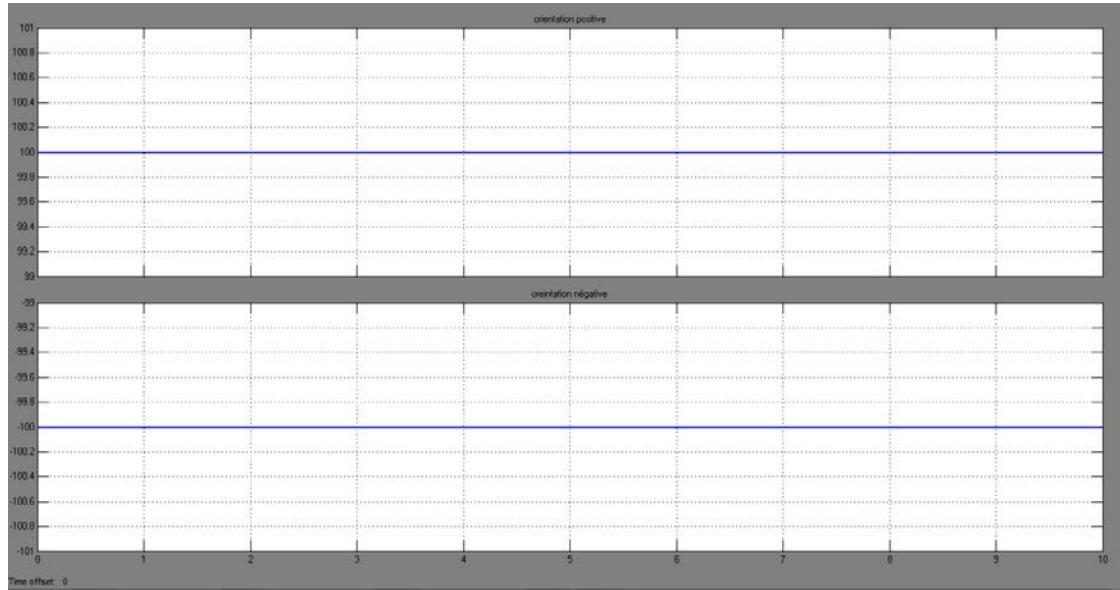


Figure 106: View of the influence of the orientation of the source of force

It can be noted that the orientation of the source has a direct influence on the direction of force exerted on the spring + damper.

2. Placing and orienting the sensors

When placing a sensor, the sign of the measurement depends on the orientation of the sensor.

Example of measuring electrical parameters:

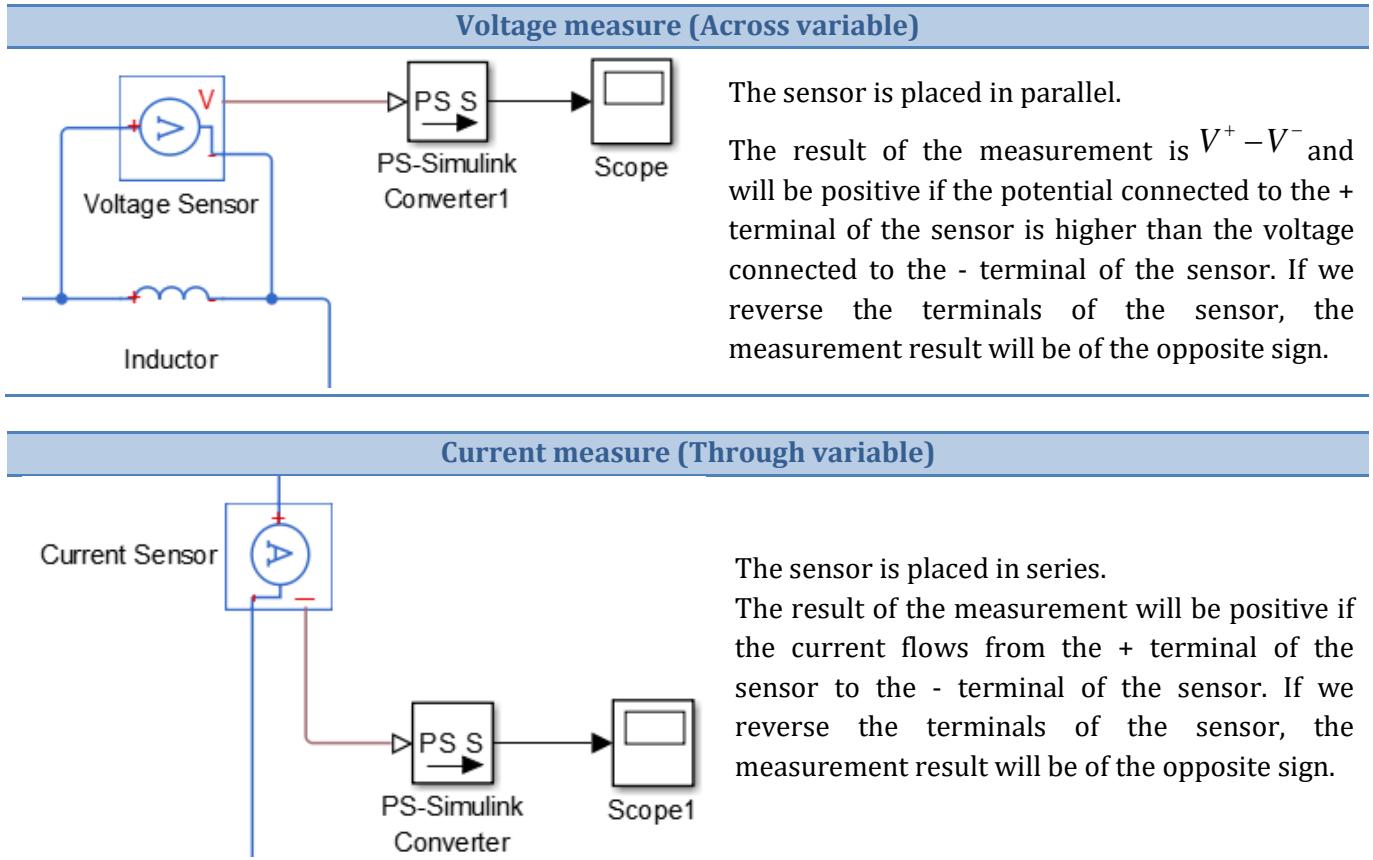


Figure 107: example of measuring electrical parameters

This reasoning can be generalized to relate to any type of sensor and any "Across" and "Through" type variable.

Open the "sensor_installation.slx" file.

This file contains the model of an RL circuit. The voltage across the coil and the current through the circuit is measured using two sensors for whose polarity has been reversed.

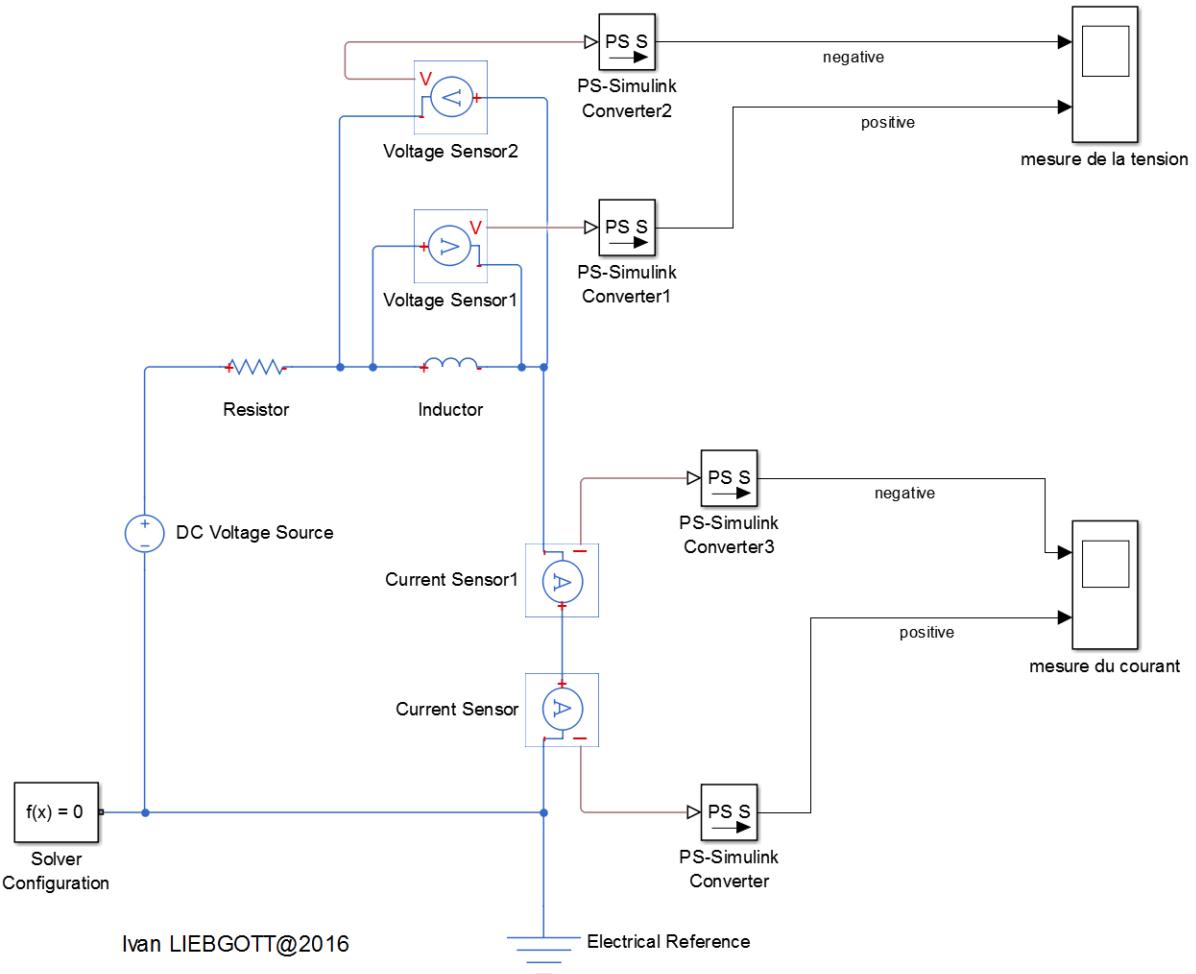


Figure 108: placing of sensors

Launch the simulation and use the scopes to observe the influence the orientation of the sensors has on the sign of the measurement obtained.

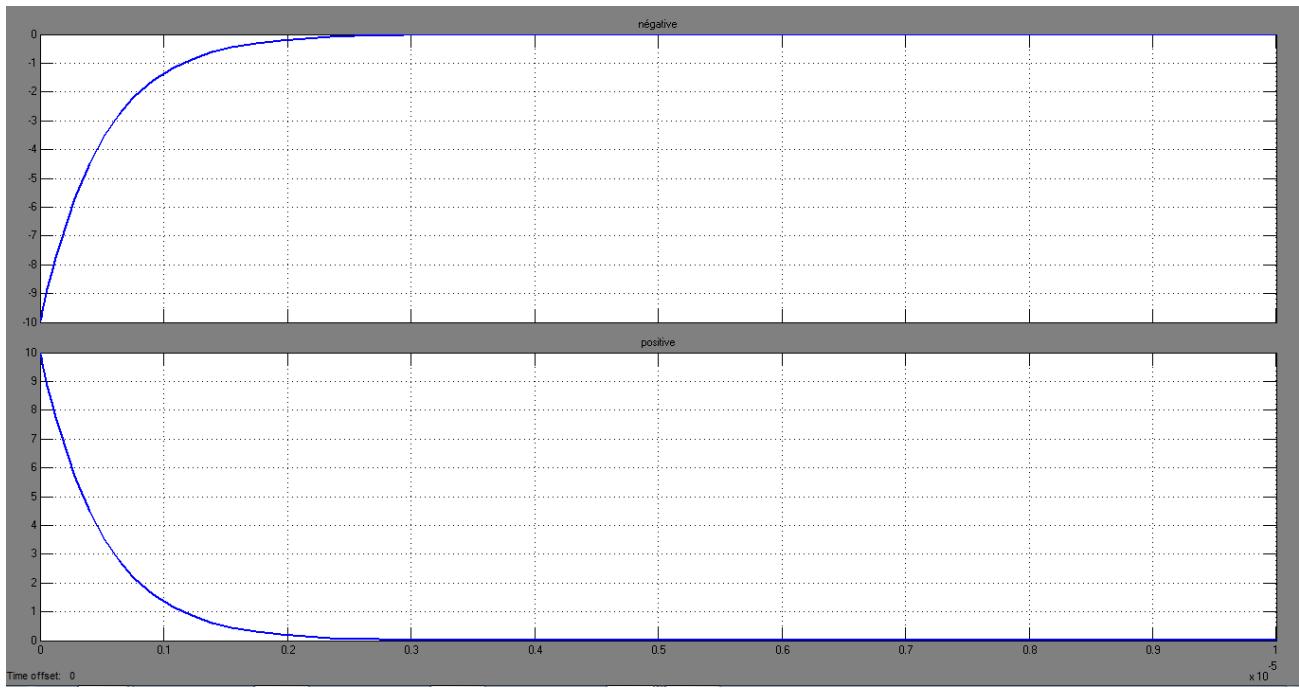


Figure 109: View of the voltage across the coil for the two sensor orientations

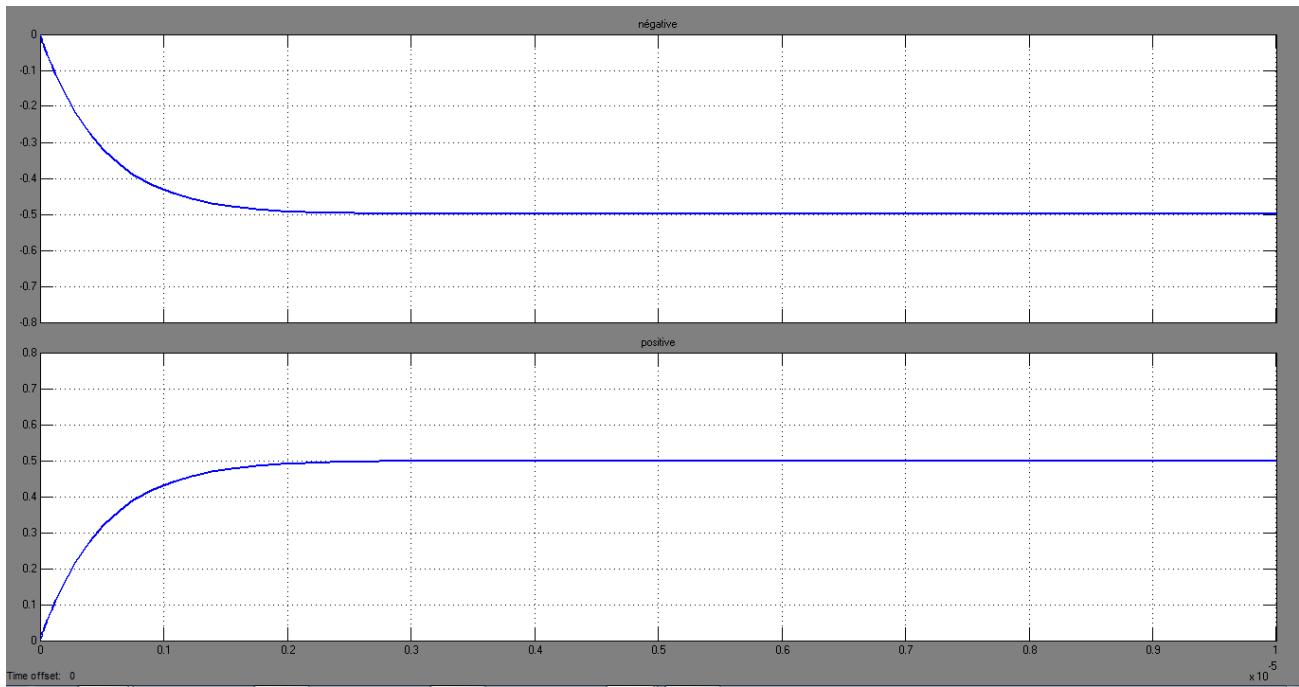


Figure 110: View of the intensity of current in the circuit according to the orientation of the sensors

The results show the influence that the orientation of the sensors has on the sign of the obtained measurement.

3. Use of components with an oriented dynamic

If using a component whose with an oriented dynamic (diode, speed reducer ...), the component must be oriented in the same way as the actual system to achieve a result consistent with the system to be modeled. If the orientation is wrong, an error message will detect this error and the modeled system will no longer be consistent with the real system.

4. Use of passive components

When using passive components (resistance, spring, dampers, piping ...), whose dynamic is not oriented, the component orientation has no influence on the behavior of the model and the observation of the "Across" and "Through" variables recorded by sensors.

5. Choice of solver

The choice of solver enables the digital resolution methods that will be implemented to be defined so that the software can complete the digital resolution of the model under the best conditions.

When the **Simscape** models become complicated, it is preferable to choose the best adapted solver for resolving this type of problem.

To do this, in the window for your model, click on **Simulation/Model Configuration Parameters** or



on the icon to display the solver settings window (Figure 111).

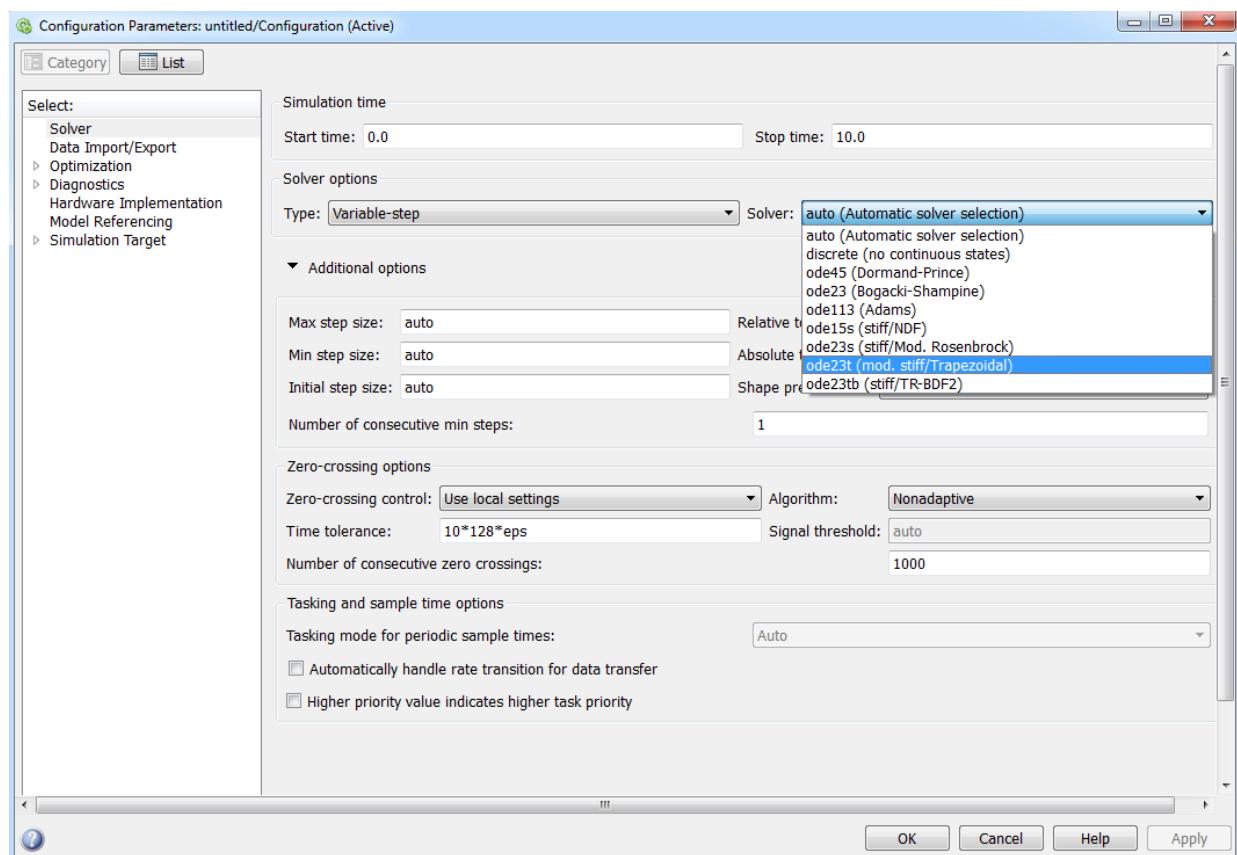


Figure 111: solver settings window

By default, the choice of solver is automatic. In general cases, this option selects the best adapted solver.

The two solvers **ode15s** and **ode23t** are the best adapted to **Simscape** and multi-physical model resolution. It is strongly advised to only work with these two solvers for **Simscape** models. If, in general cases, the automatic choice of solver does not give satisfactory results, the **ode15s** solver can be selected. If this still does not resolve the model or the calculation time is too long, **ode23t** may be used as an alternative.

In this window it also possible to specify a minimum and/or a maximum calculation step to choose the number of points and the resolution of the calculation. We then complete the **Max step size** and **Min step size** fields.

6. Problems which the solver may encounter

At the launch of the simulation, the solver will digitally solve the model to determine the evolution of all variables over time. This very complex mathematical process will be automatically handled by the software. However, problems can occur which stop the calculation process. This results in an error message indicating that the solver cannot reach the end of the resolution.

The solvers used by MATLAB are extremely robust, in most cases any problems result from an error in the modeling. The model should be reexamined and any potential errors corrected. When conducting a complex model, it will simulate each part of the model separately to identify the parts of the model that are causing errors.

Here are some tips to help solve the most common problems:

- The “Solver” block is missing;
- The reference for a physical field that has been used is missing;
- Components are missing or the components have not been configured with valid values. The quantities calculated by the solver have no physical sense (infinite speed, infinite pressure, temperature below absolute zero ...). In this case, the solver will stop the calculation and indicate the problem it has encountered. The configuration or position of the components should be reexamined.

IV. Examples of multi-field modeling

This section provides, through examples, an overview of the most common physical fields that can be addressed with **Simscape**.

Each example proposes the construction of a model and uses the results of the simulation. Important concepts are introduced as needed during the construction of the models in order to acquire the skills necessary for multi-physical models using **Simscape**. You are therefore advised to work through all the examples in this section.

A. Electro-mechanical field - Linear axis

The system being modeled is a linear axis. The actuator is a DC gear motor controlled by a variable voltage source. The rotational movement is converted to translational movement by a pulley/belt system. The objective is to evaluate responses in speed and axis position to estimate the response time and speed or any other type of performance. It will also be possible to measure the electrical parameters such as motor current.



Figure 112: photo of the linear axis

The system studied involves 3 physical fields:

- Electrical field
- Mechanical rotation field
- Mechanical translation field

Some components belong to a particular physical field and other components can make the conversion from one to another of the physical fields. This is the case with the “**DC Motor**” component which will convert from the electrical field to the mechanical rotation field and the “**Wheel and Axle**” component which will convert from the mechanical rotation field to the mechanical translation field.

1. Choice of components

Only components that have not yet been used previously in this document are referenced in the table shown in Figure 113.

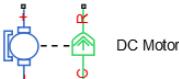
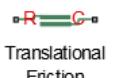
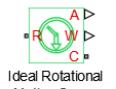
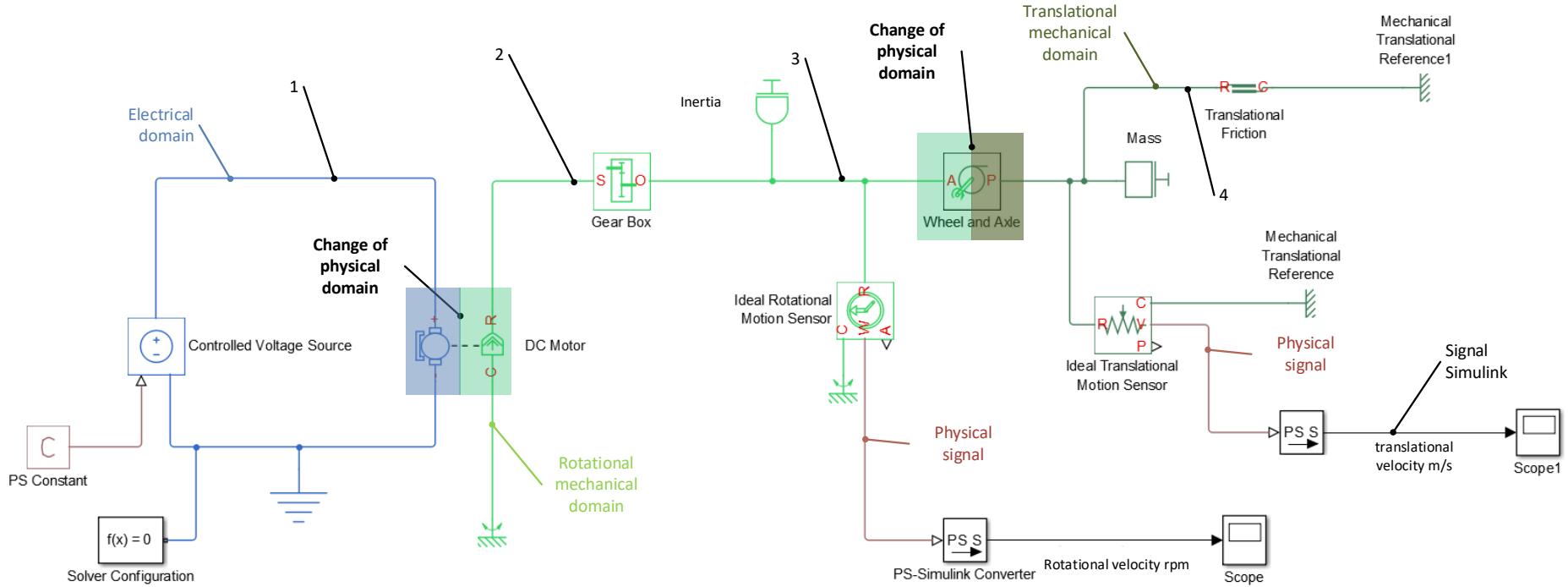
Component function	View	Library
Source of variable voltage	 Controlled Voltage Source	Simscape/Foundation Library/Electrical/Electrical Sources
Physical signal source	 PS Constant	Simscape/Foundation Library/Physical Signals/Sources
DC motor	 DC Motor	Simscape/SimElectronics/Actuators and drivers/Rotational Actuators
Reducer	 Gear Box	Simscape/Fondation Library/Mechanical/Mechanisms
Inertia	 Inertia	Simscape/Fondation Library/Mechanical/Rotational Elements
Conversion of rotational movement in to translational movement	 Wheel and Axle	Simscape/Fondation Library/Mechanical/Mechanisms
Mass	 Mass	Simscape/Fondation Library/Mechanical/Translational Elements
Dry and viscous friction in translation	 Translational Friction	Simscape/Fondation Library/Mechanical/Translational Elements
Position or translation speed sensor	 Ideal Translational Motion Sensor	Simscape/Fondation Library/Mechanical/Mechanical Sensors
Position or translation speed sensor	 Ideal Rotational Motion Sensor	Simscape/Fondation Library/Mechanical/Mechanical Sensors

Figure 113: the components required for modeling the linear axis in Simscape

The Simscape model of the linear axis is represented in Figure 114 and enables the various physical fields that involved in the model to be visualized, as well as the components which create the conversions between physical fields. The interaction between the connections of the different physical fields and the associated actual components is provided.



Correspondance physique des connexions:

- 1: electrical wire (electrical domain)
- 2: motor shaft (rotational mechanical domain)
- 3: gear box shaft (rotational mechanical domain)
- 4: belt (translational mechanical domain)

Figure 114: view of the physical fields involved in modeling the linear axis with Simscape

2. Placement and assembly of components

In the **Simulink Library Browser** window, execute the **File/New/Model** command to create a new file.

Slide/Move the various blocks from the libraries, place them in the working window, and link them in order to obtain the configuration shown in Figure 115.

In Figure 116 there are some useful commands for the layout of a model.

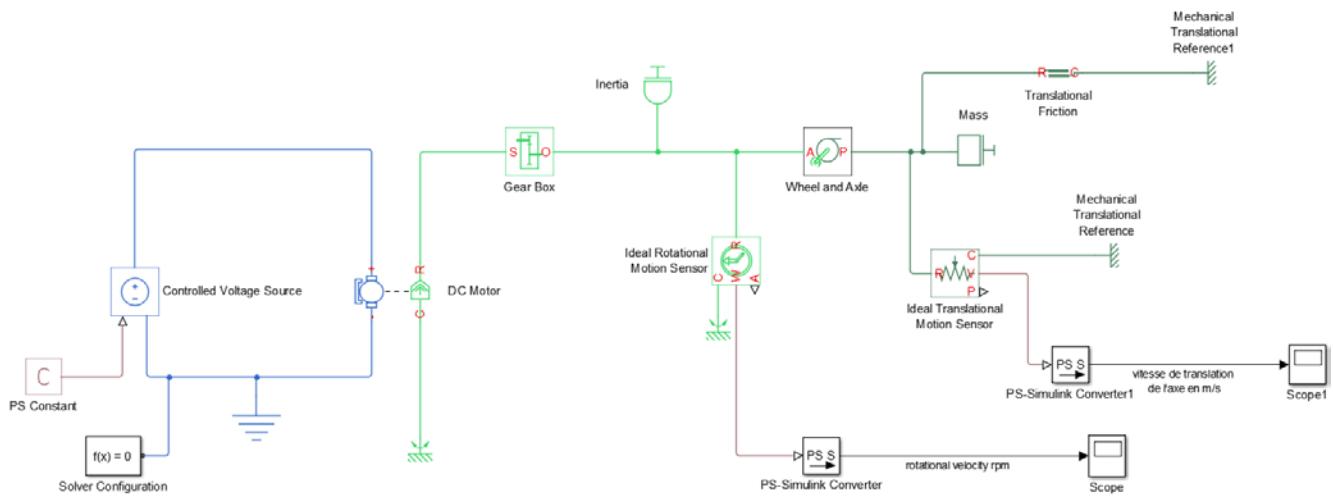


Figure 115: Simscape model of the linear axis

Useful commands	
Functions	Actions
Give a name to a Simulink signal	Right-click directly on the Simulink signal connection, select Properties , then complete the Signal Name field in the window. The signal name will appear in the scope key.
Rename a block	Double-click directly on the name of the component then type in the new name (caution - two components cannot have the same name)
Deleting a component name	Right-click on the component, select Format then deselect Show Block Name

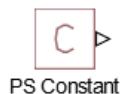
Figure 116: useful commands

3. Configuration of components

Perform the configuration of various blocks in accordance with the information provided below.

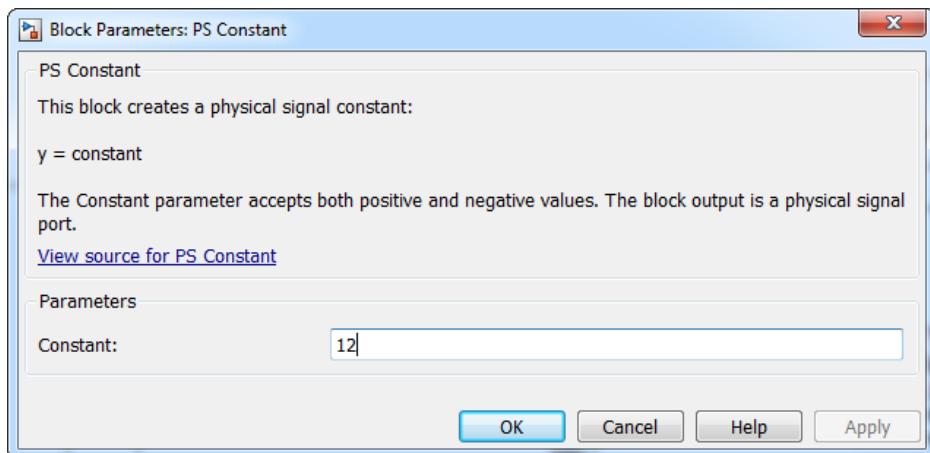
Configuration

Constant PS



Simscape/Foundation Library/Physical Signals/Sources

The value in this component sets the value of the control voltage in the variable voltage source.
Configuration of this component is set by entering the physical signal value, here 12V.



Configuration

CONTROL VOLTAGE SOURCE

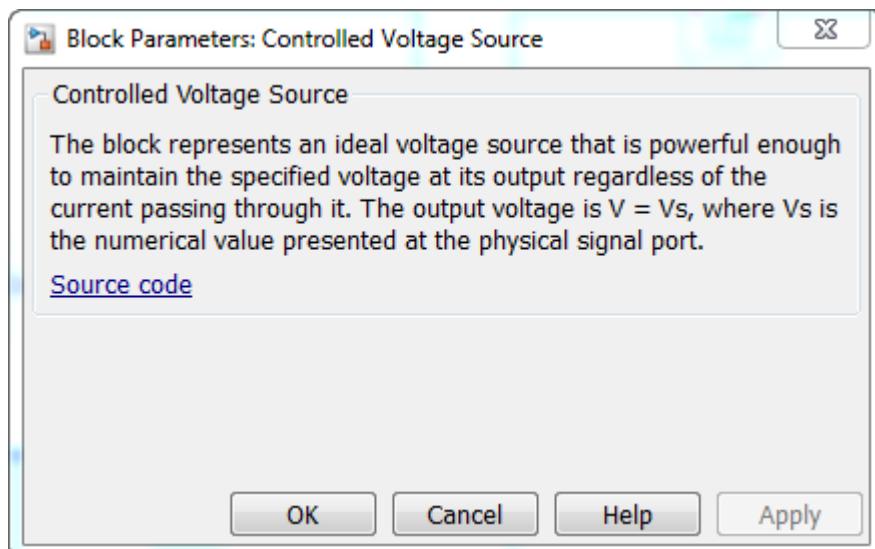


Controlled Voltage Source

Simscape/Foundation Library/Electrical/Electrical Element

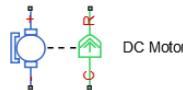
This component does not require any configuration.

The port corresponding to the physical signal indicates the value of the voltage imposed on the source terminal at the variable voltage source.



Configuration

DC MOTOR

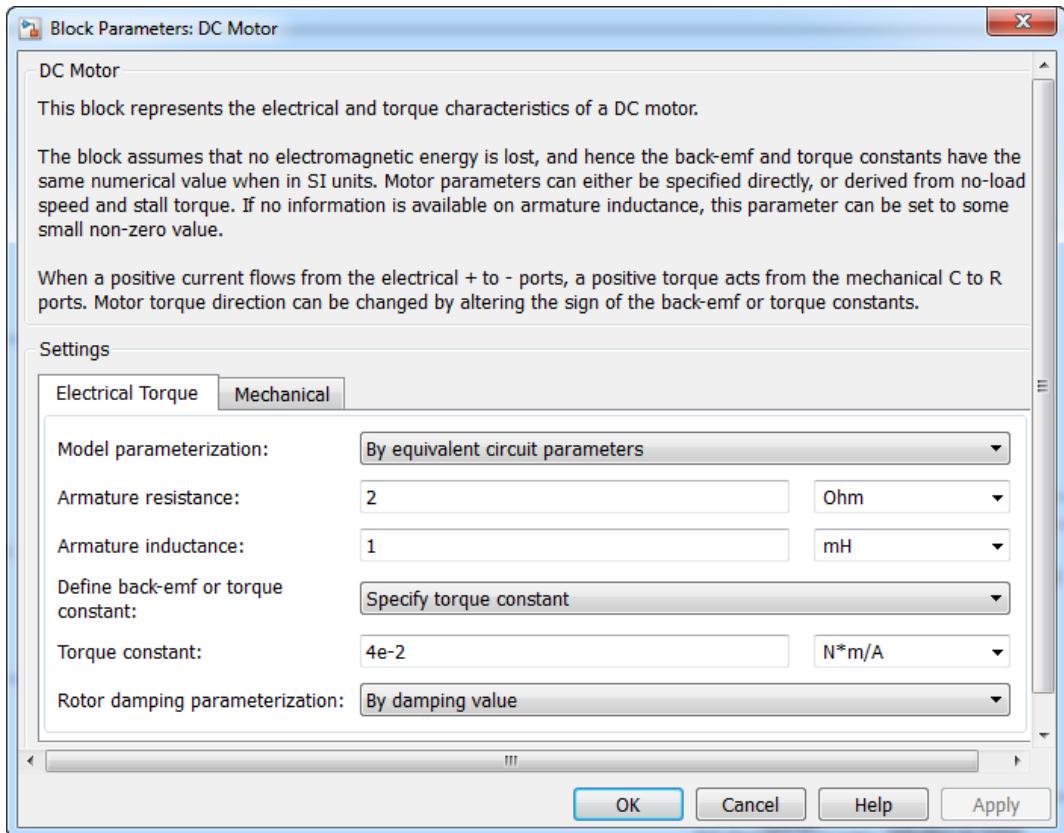


Simscape/SimElectronics/Actuators and drivers/
Rotational Actuators

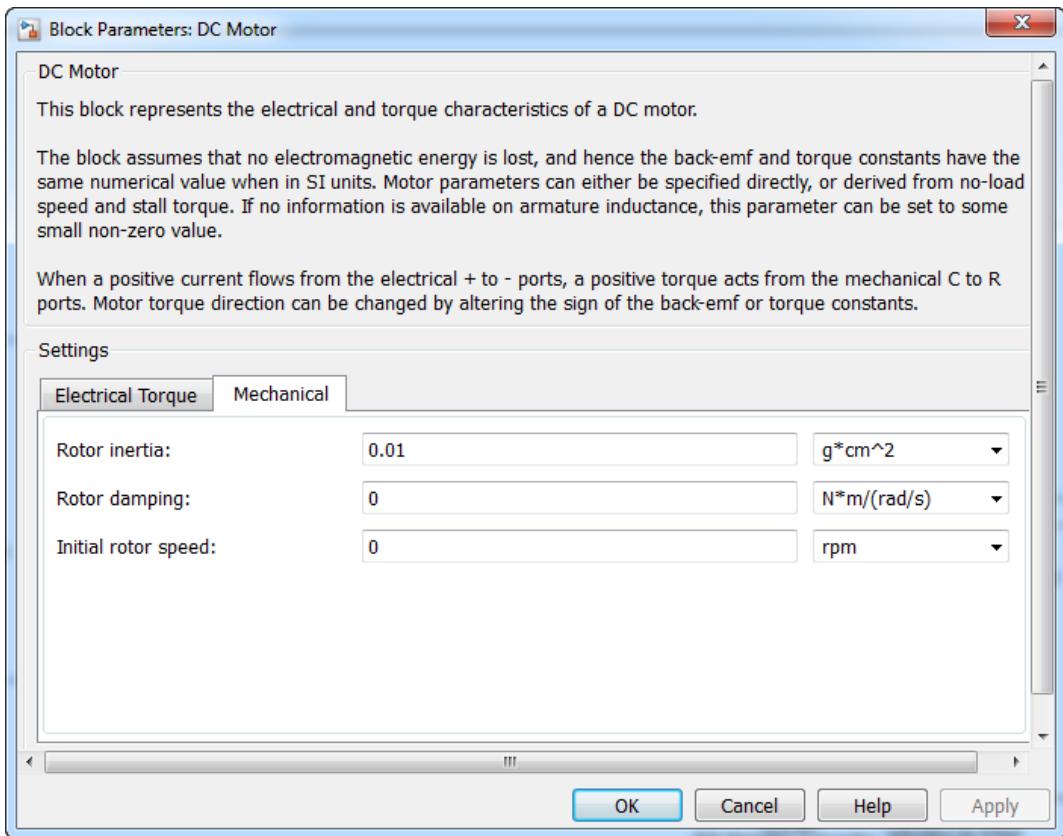
This component models a DC motor. It has 4 **PCP** ports, 2 in the electrical field (+ and -) and 2 ports (**C** and **R**) in the mechanical rotation field. The PCP in the mechanical field, **C** (Cage, fixed part) will be connected to a mechanical rotation reference that can be associated with the frame. The other mechanical port type **R** (Rotor, movable part) will be associated with the output shaft of the engine.

With this block we can use a DC motor, already modeled, and configure it using manufacturer data. Two tabs are available for entering electrical and mechanical configurations.

Electrical Torque tab



Mechanical tab



By default, the **Model Parameterization** under the **Electrical Torque** tab is set using **By equivalent circuit parameters**. In accordance with the available information in the motor constructor file, it will be possible to modify the type of parameters to set, by choosing **By stall Torque & and no-load speed** or **By rated power, rated speed & noload speed**.

Configuration

GEAR BOX

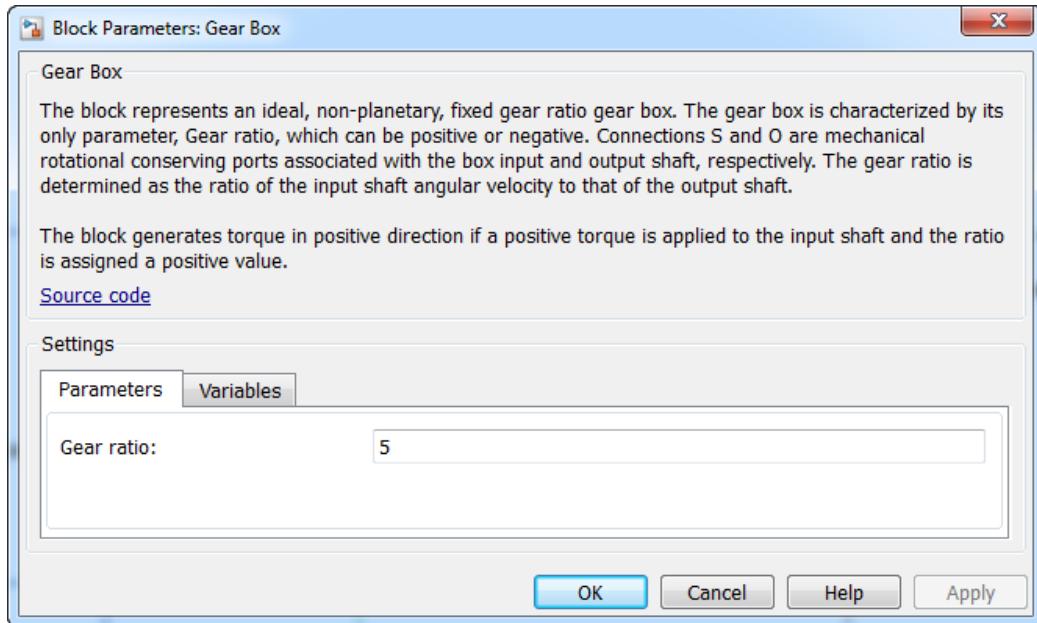


Simscape/Foundation
Library/Mechanical/Mechanisms

This component enables modeling of perfect reducer behavior.

By convention the Gear ratio is defined as $Gear\ ratio = \frac{\omega_{entrée}}{\omega_{sortie}}$ which means that for a reduction ratio

$$r = \frac{1}{5} = \frac{\omega_{sortie}}{\omega_{entrée}}$$
 the Gear ratio specified will be equal to 5.



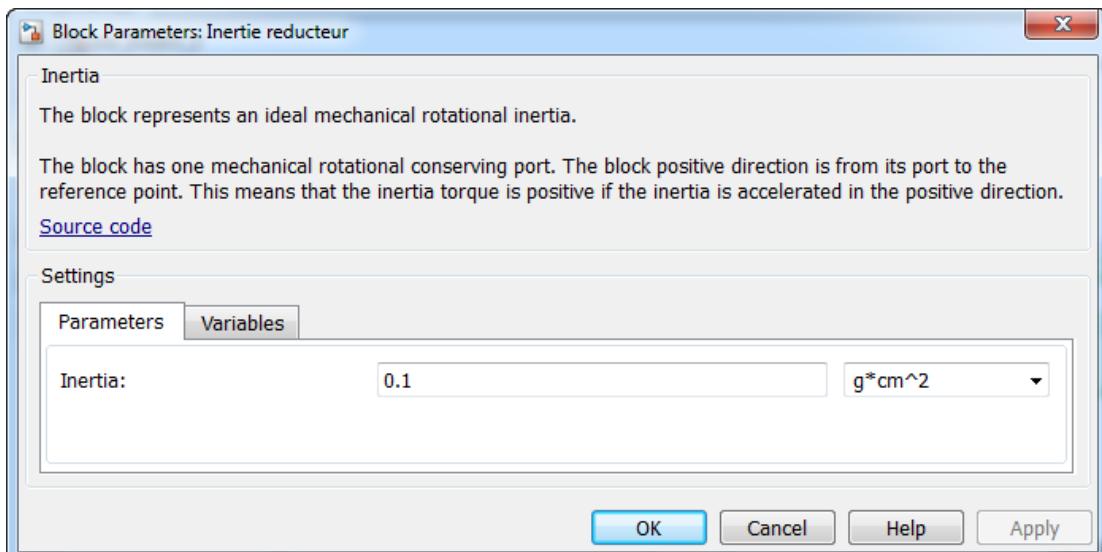
Configuration

INERTIA



Simscape/Foundation
Library/Mechanical/Rotational Elements

This component enables an inertia associated with a connection in the mechanical rotation field to be modeled.



It is also possible to specify an initial speed for this inertia. Acausal modeling allows for non zero initial settings (in contrast with a causal approach where the transfer or physical masses must be nil at t=0).

Configuration

WHEEL AND AXLE

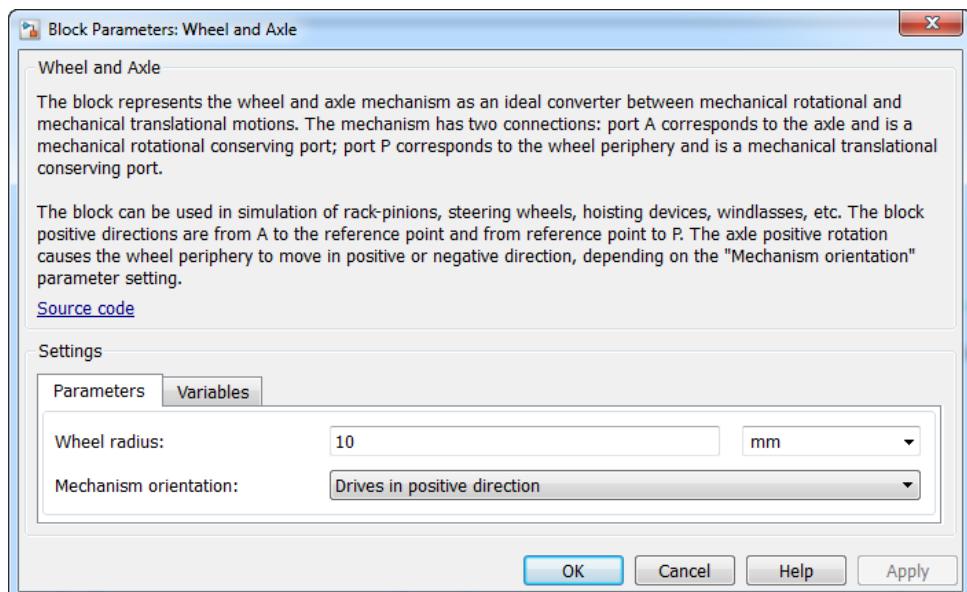


Simscape/Foundation Library/Mechanical/Mechanisms

This component can model a perfect power conversion between a rotational movement and a translation movement.

The parameter to set is the **radius** of the pulley.

It is also possible to specify a sign convention to carry out this conversion.



Configuration

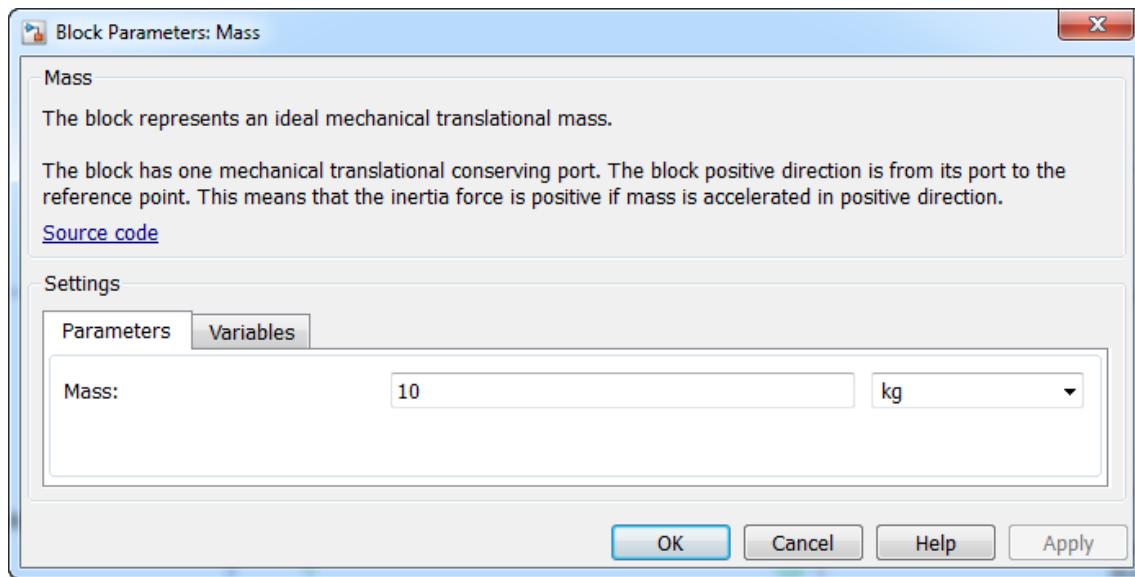
MASS



Simscape/Foundation Library/Mechanical/Translational Elements

This component allows for modeling a mass associated with a connection in the mechanical translation field.

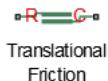
Here, the mass corresponds to the mass of the carriage driven in translation by the pulley.



It is also possible to specify an initial speed for this mass.

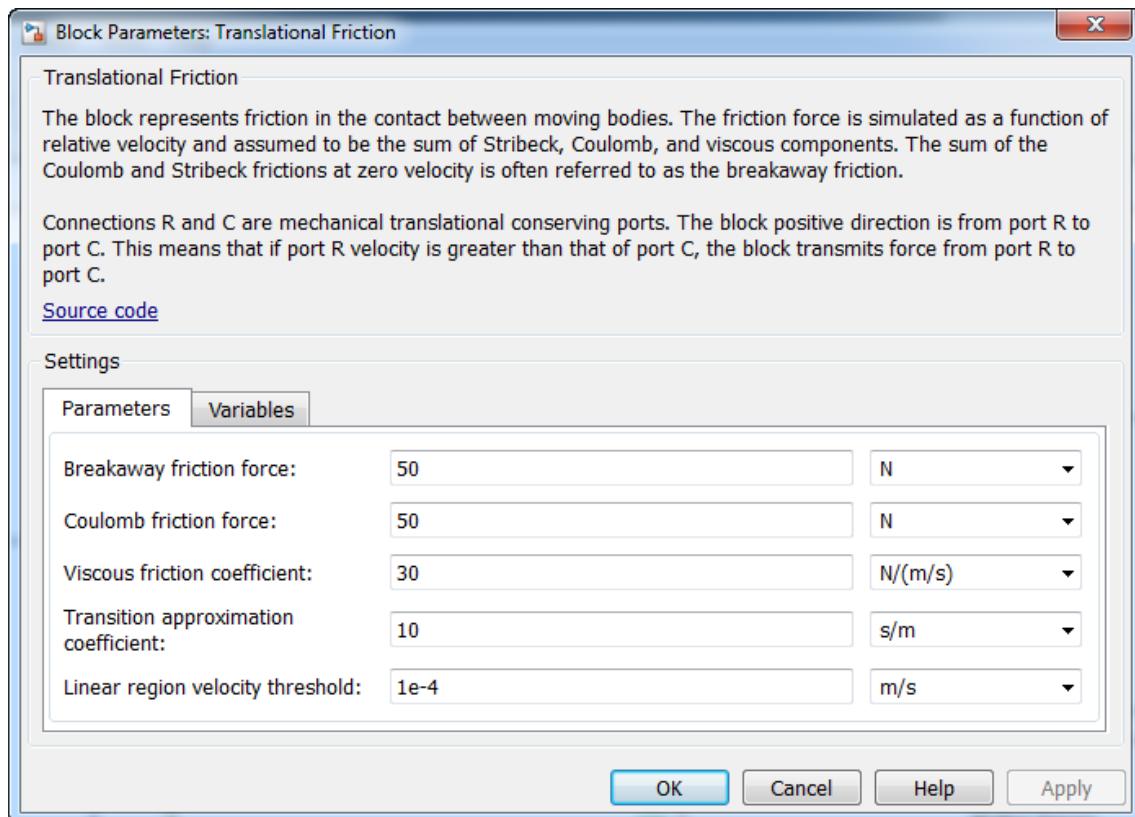
Configuration

TRANSLATIONAL FRICTION



Simscape/Foundation Library/
Mechanical/Translational Elements

This component enables a friction force (dry or viscous) to be which acts on an element in the mechanical translation field to be modeled. Port **C** is connected to the frame and port **R** to the connection that is undergoing the friction force. The force will be exerted by the frame on the element of the mechanical translation field with which port **R** is connected. The force always acts in opposition to the movement. Here there is a force exerted by the frame on the carriage, which will act as a disturbance to the system.



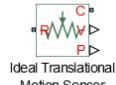
Breakaway friction force: a force which limits adhesion (force from where the movement starts)

Coulomb friction force: a friction force (which occurs during movement)

Viscous friction coefficient: coefficient of viscous friction

Configuration

IDEAL TRANSLATIONAL
MOTION SENSOR

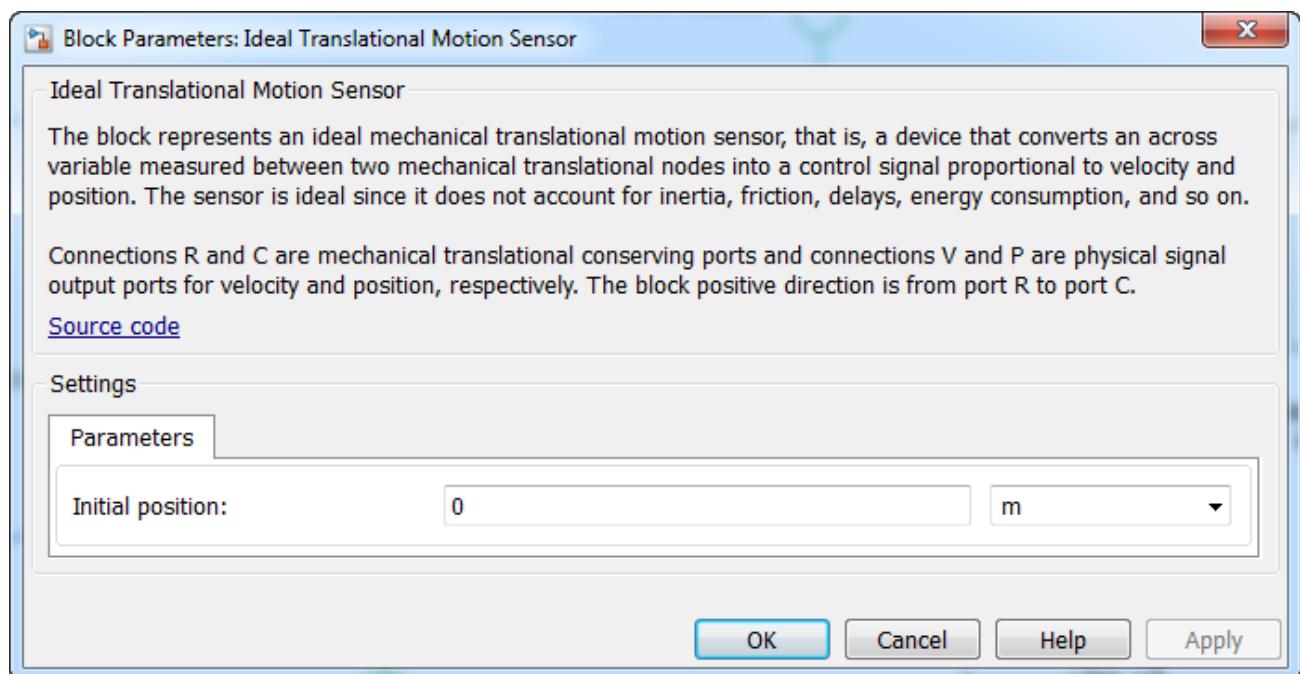


Simscape/Foundation
Library/Mechanical/Mechanical Sensors

This sensor enables measurement of translation speed ("Across" type variable) or a position in translation. It therefore increases in parallel. Port **R** is connected to the connection for which you want to know the speed relative to port **C**, which is connected to the chosen reference. The physical signal you can read is either the translation speed (port **V**) or the position (port **P**).

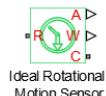
Here, we want to know the movement speed of the carriage relative to the frame.

It is also possible to specify the initial value of the measurement at the start of the simulation (0 by default).



Configuration

IDEAL ROTATIONAL MOTION
SENSOR

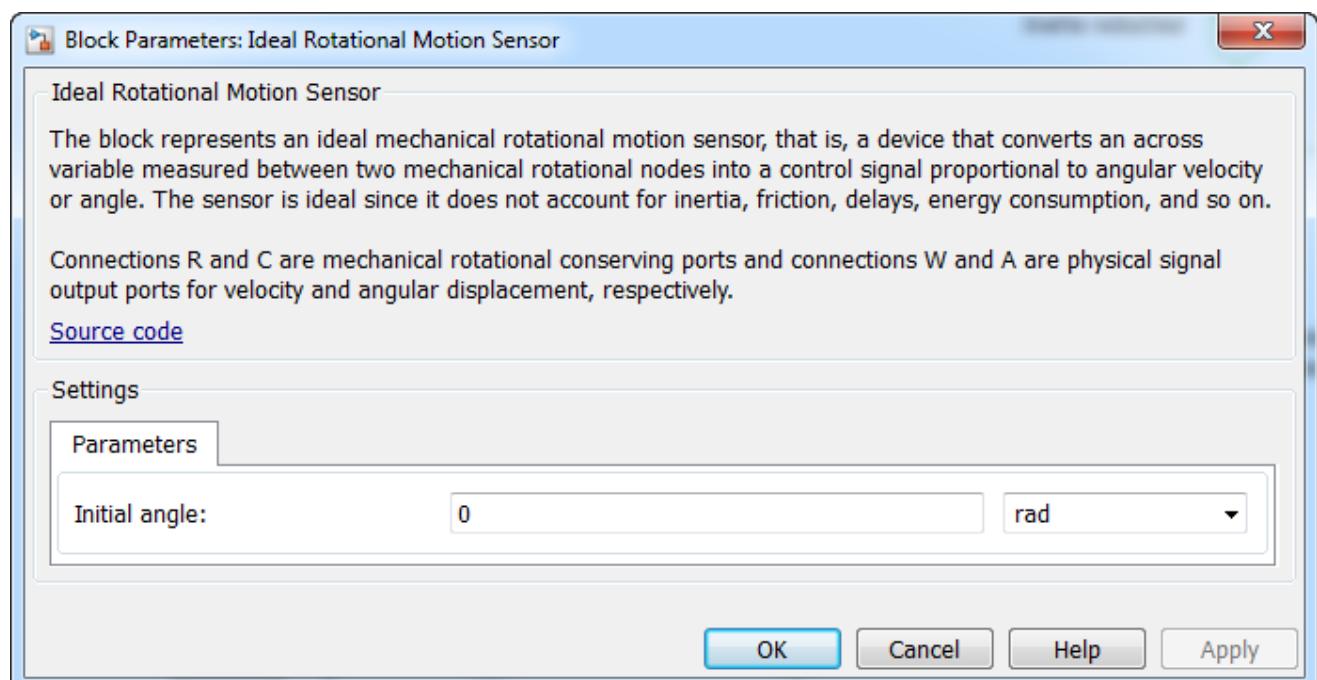


Simscape/Foundation
Library/Mechanical/Rotational Sensors

This sensor enables the measurement of rotational speed ("Across" type variable) or an angle. It therefore increases in parallel. Port **R** is connected to the connection for which you want to know the speed relative to port **C**, which is connected to the chosen reference. The physical signal that can be read is either the rotational speed (port **W**) or the angle (port **A**).

Here, we want to know the movement speed of the carriage relative to the frame.

It is also possible to specify the initial value of the measurement at the start of the simulation (0 by default).



Configuration

PS-SIMULINK CONVERTER



Simscape/Utilities

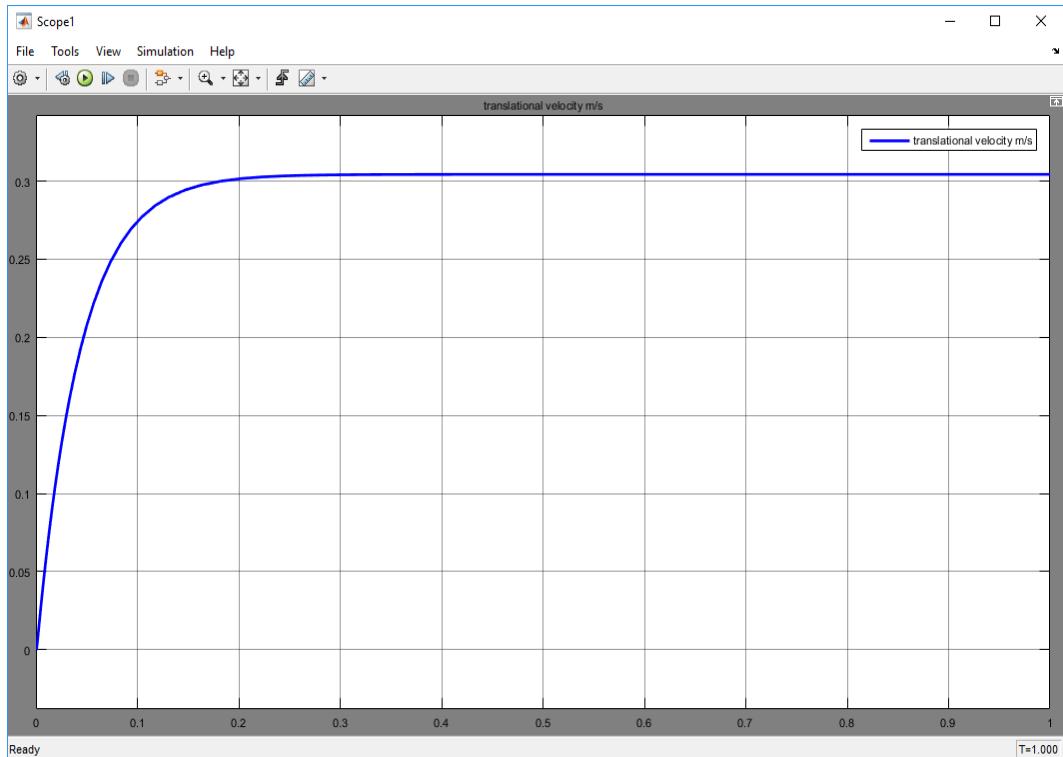
This component enables the conversion of a physical image of the parameters measured by the Simulink signal so that it can be displayed in a scope. Specify that the translation speed of the carriage is recorded in m/s and the rotational speed of the shaft in turns/min (rpm in Simscape)

The file containing the configured model is available under the name **linear_axis_0_US.slx**

4. Open loop model simulation

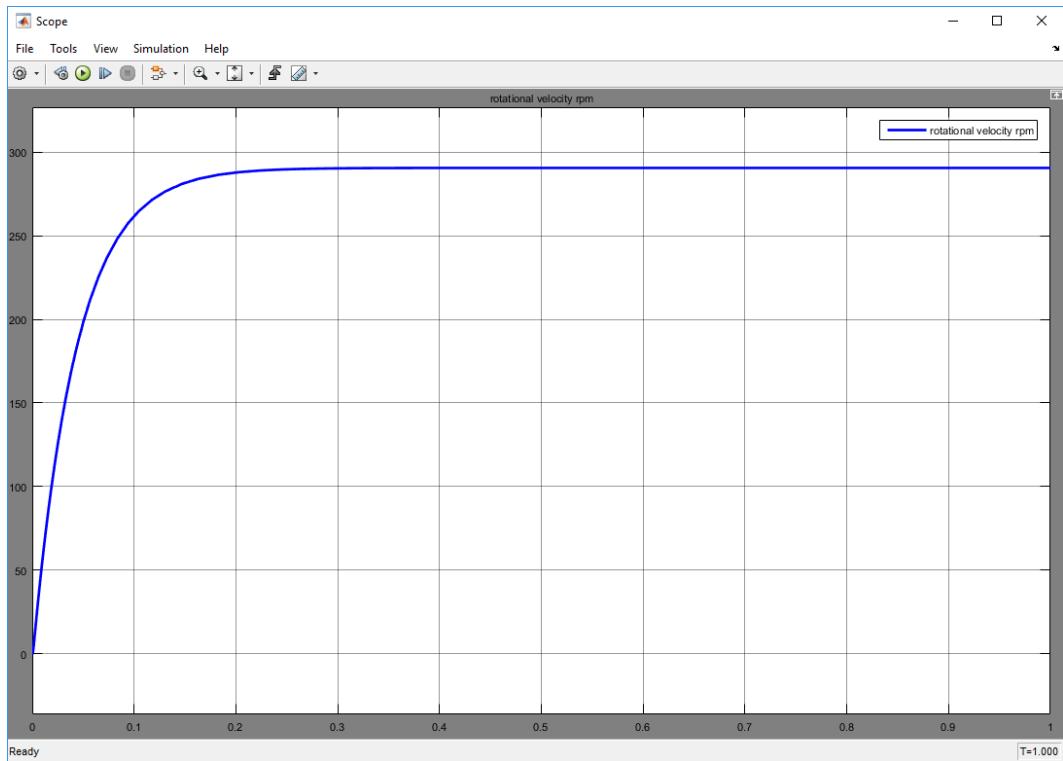
Launch the simulation, specifying a simulation time of 1s.

Observe the translation speed by double-clicking on the scope.



The translation speed of the carriage stabilizes at around 0.3 m/s

Observe the rotational speed of the tree at the exit of the reducer by double-clicking on the scope.



The rotational speed of the shaft at the exit of the reducer stabilizes at around 290 tr/min.

5. Using the Simscape Data-logger

When designing a model, it is very useful to have access to all the variables involved in the model. The first solution may consist of positioning sensors wherever variables need to be observed. This method will quickly become long and tedious and make the model very difficult to read.

To meet this design need, **Simscape** offers the option of storing the evolution of all the variables that change within the model. It is therefore possible, with the help of the **Simscape** Data-logger, to view the evolution of all the variables.

The function that creates data viewing must be accessible in the MATLAB “path”. This function is called “**ssc_explore.m**” and receives the variable that contains all the data of the simulation as an argument.

The “**Simscape_logger**” file which contains this function must be in the **MATLAB** “path” so that the “**ssc_explore.m**” function is accessible from the command window. The “**Simscape_logger**” file is included in the “**Modélisation_multi_physique_modeles/ Chapitre_2_Simscape**” folder which is already in place in the **MATLAB** path.

You now need to specify the name of the variable which will store all the simulation data before using the “**ssc_explore.m**” function.

Click on **Simulation/Model Configuration Parameters**, then choose the **Simscape** tab from the left side of the window.

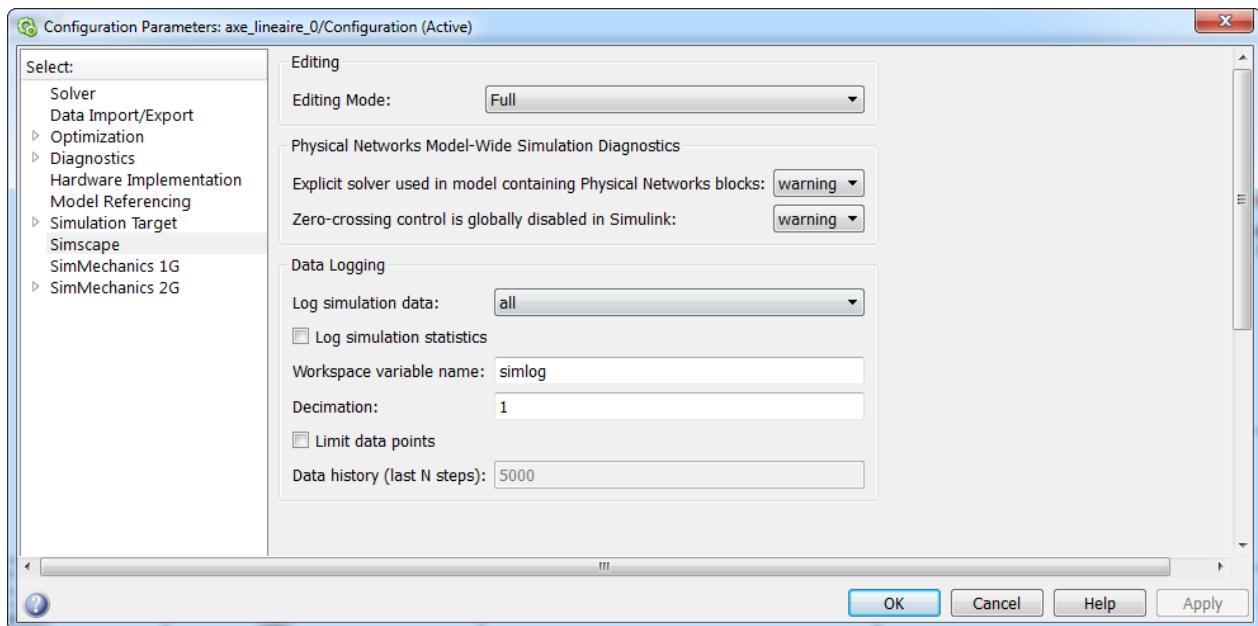


Figure 117: specification of the Simscape variables to be taken into account in the logger

Choose **all** in the **Log simulation data** menu.

Choose the name of the variable which will store all the simulation data in the field **Workspace Variable Name** (here **simlog**, but this name can be modified)

Deselect **Limit data points** to ensure that you keep all the simulation data.

Relaunch the simulation.

Return to the MATLAB command window and type **ssc_explore(simlog)**

The **Simscape** logger will be displayed and all the simulation data may be viewed.

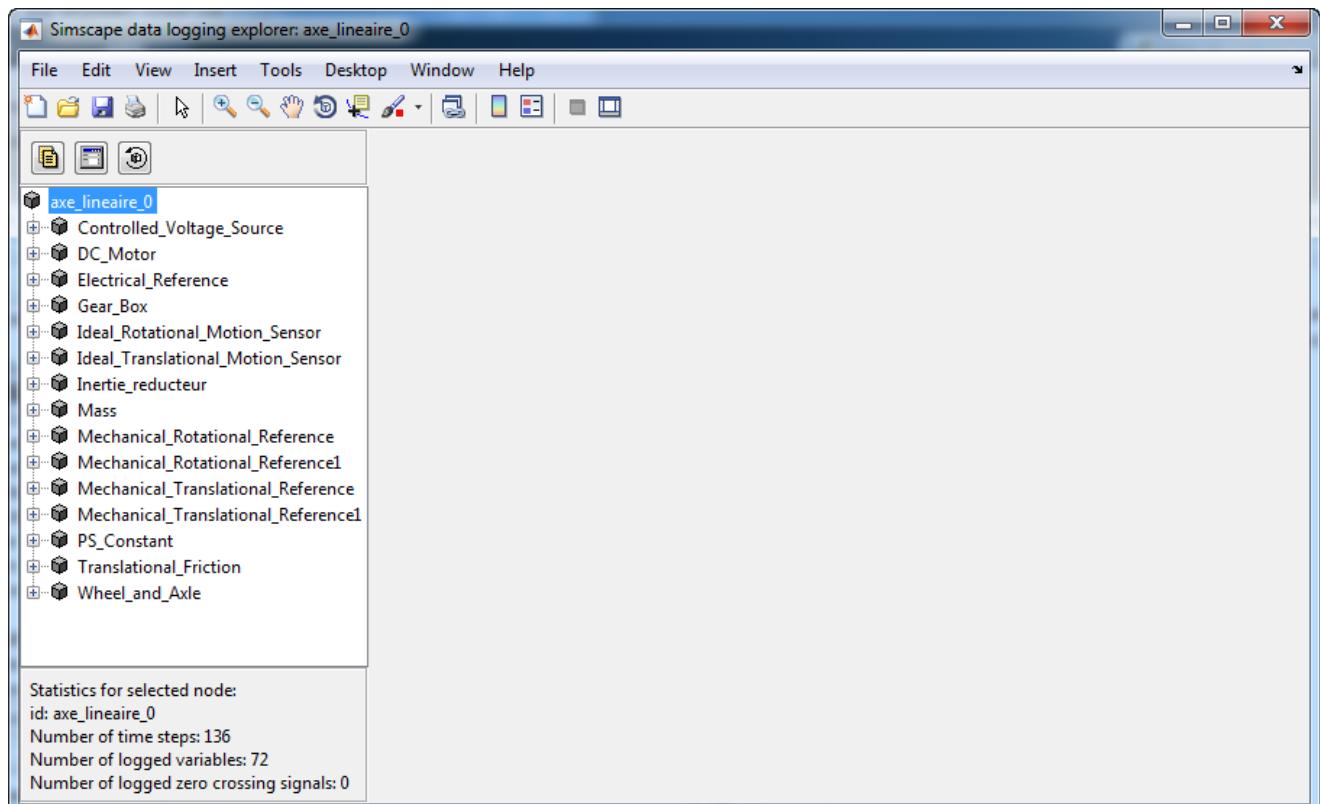
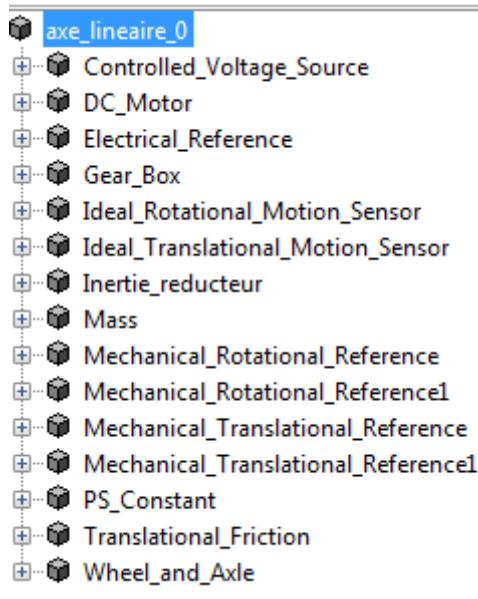


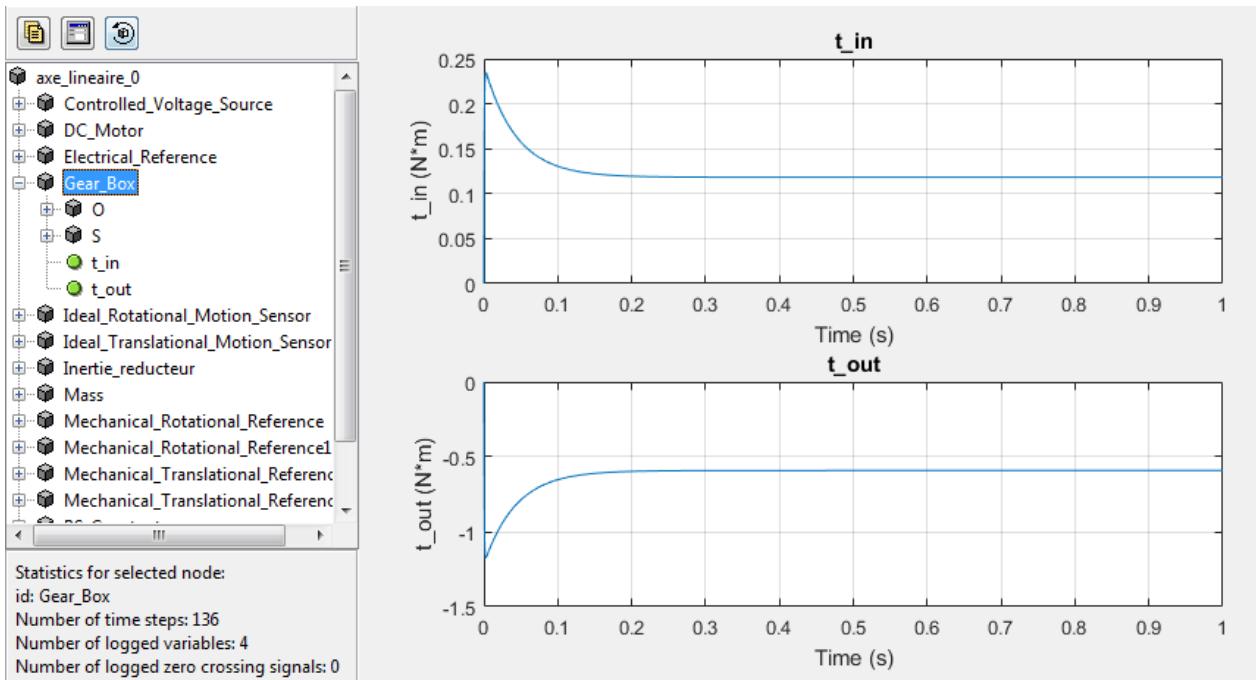
Figure 118: Simscape logger window

The names of the model components will be displayed on the left side of the window.

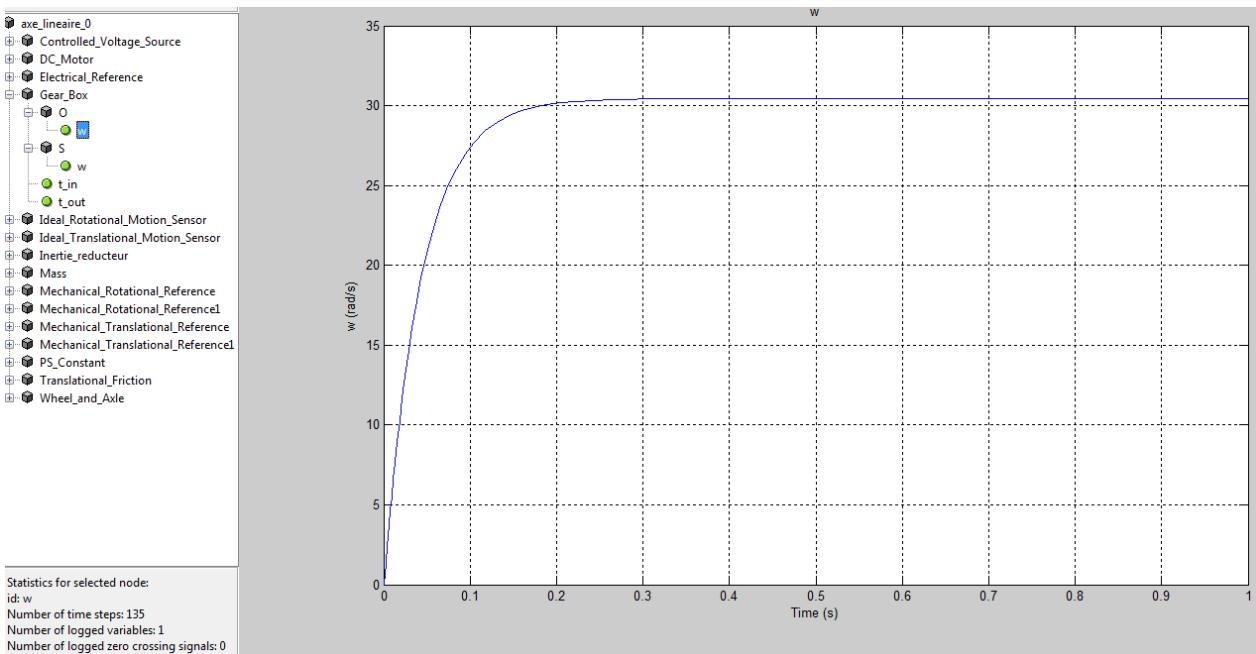


Choose, for example, the **Gear_Box** component and expand the tree to show all the variables of the component. The tree will show the two ports of the component, **O** and **S**.

Two curves will appear showing the evolution of the **Moment** in N.m ("Through" type variable), with **S** at the input and **O** at the output of the component.



It is also possible to access the rotational speed (“Across” type variable) at the entrance **O** and exit **S** level of the component.



The principle of the Data-logger is to be able to observe the “Across” and “Through” type variables for all the model components. The **PCP** type ports appear on the tree for every component. Simply click on the variable to observe the curve.

The **Simscape** logger will quickly become essential and saves a considerable amount of time when creating new models.

6. Creation of sub-systems

Sub-systems can be created to help simplify the understanding and exploration of the model. This process will quickly become indispensable for creating of complex models.
The principle consists of selecting a group of components to create a sub-system.

We are going to create a sub-system which includes the motor and its power supply.

Select as shown in **Erreur ! Source du renvoi introuvable.** the components which will make up the sub-system and slide the mouse onto the **Create Subsystem** icon, bottom right in the selection.

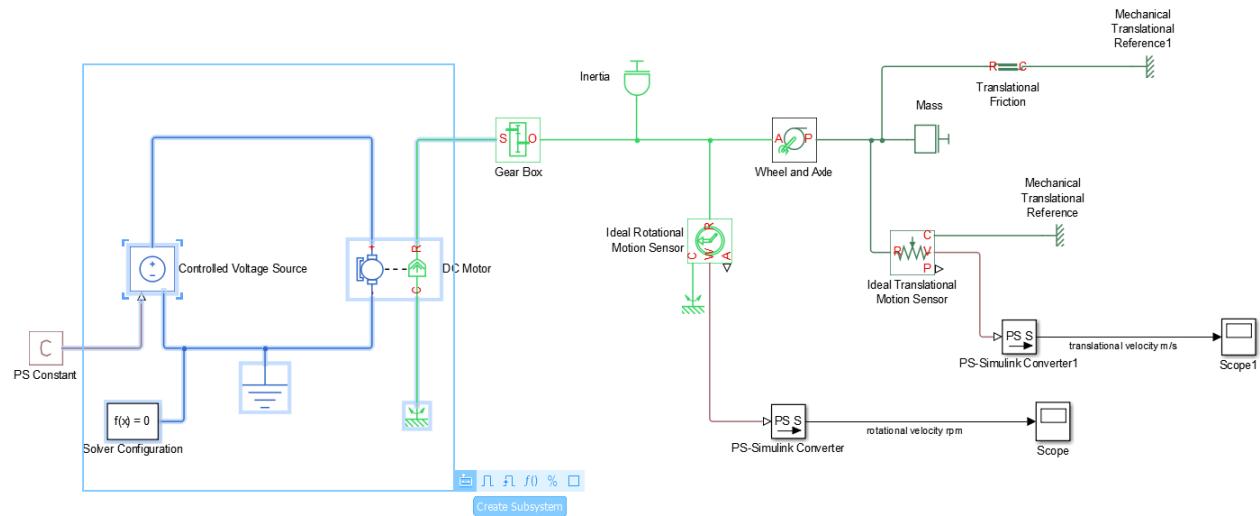


Figure 119: creation of a sub-system

It is also possible to **right-click** on the selection and choose **Create Subsystem from Selection** in the dropdown menu or use the keyboard shortcut **Ctrl+G**.

Resize the sub-system if necessary to see ports **Conn1** and **Conn2** of the component.

Useful commands	
Functions	Actions
Resize a component	Select the component with the left button on the mouse, than drag the handles which appear around the component to resize it.

Figure 120: useful commands

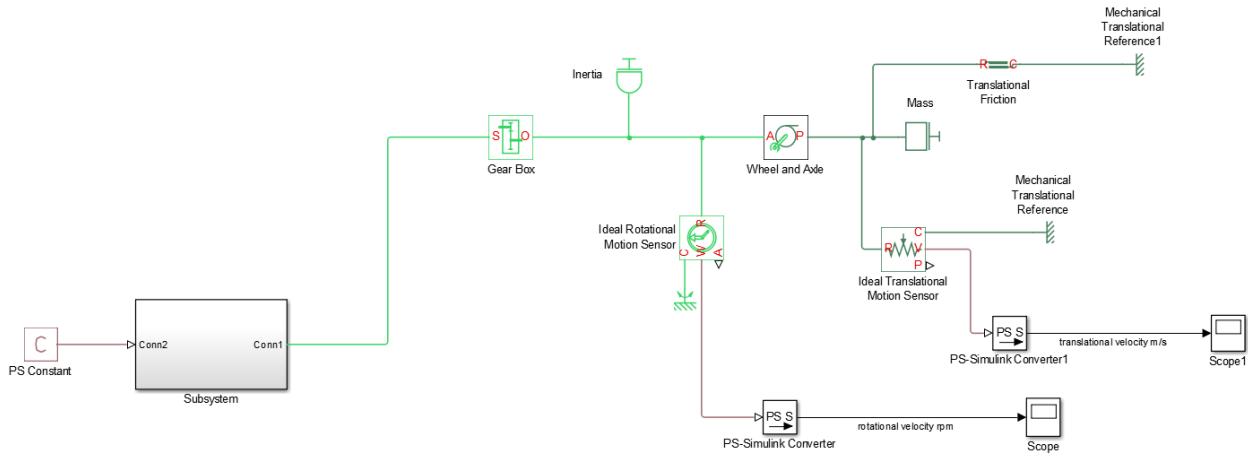


Figure 121: creation of a sub-system

For information, it is also possible to return to the previous model size by “turning off” the sub-system. To do this, right-click on the sub-system and choose the command **Subsystem & Model Reference/Expand Subsystem**.

Double click on the name of the sub-system situated in the sub-system frame to replace it with “**Motor**”

Then **double-click** on the sub-system to see which components make it up.

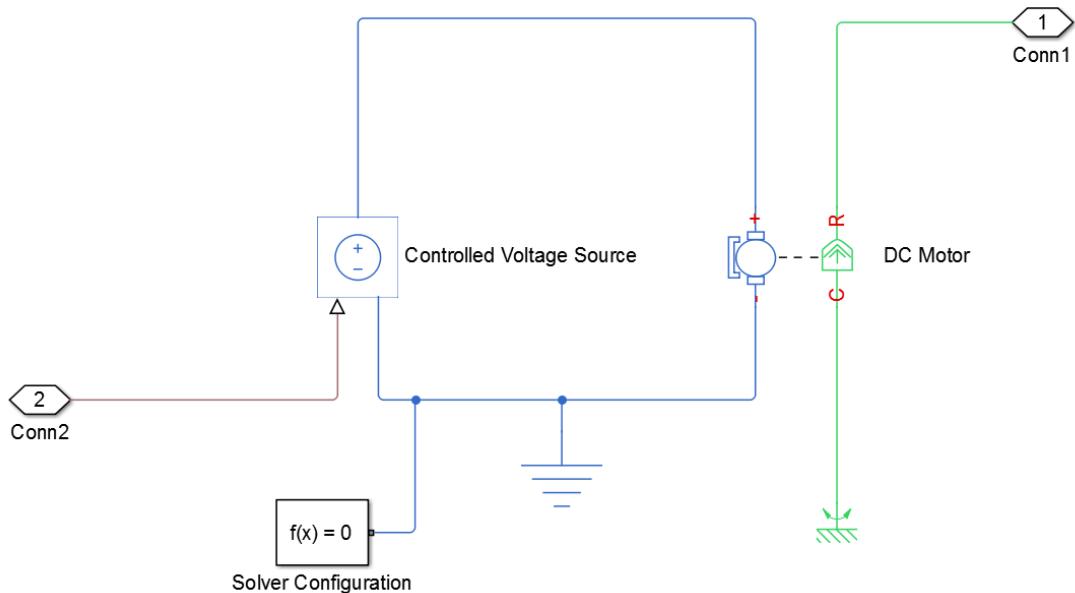


Figure 122: View of the contents of the sub-system

You can observe the external connectors with ports **Conn1** and **Conn2**.

It is also possible to rename these ports.

Double-click directly on the name of port **Conn2** and replace it with “**Control voltage**”.

Do the same by replacing **Conn1** with “**Motor Torque**” (Figure 123)

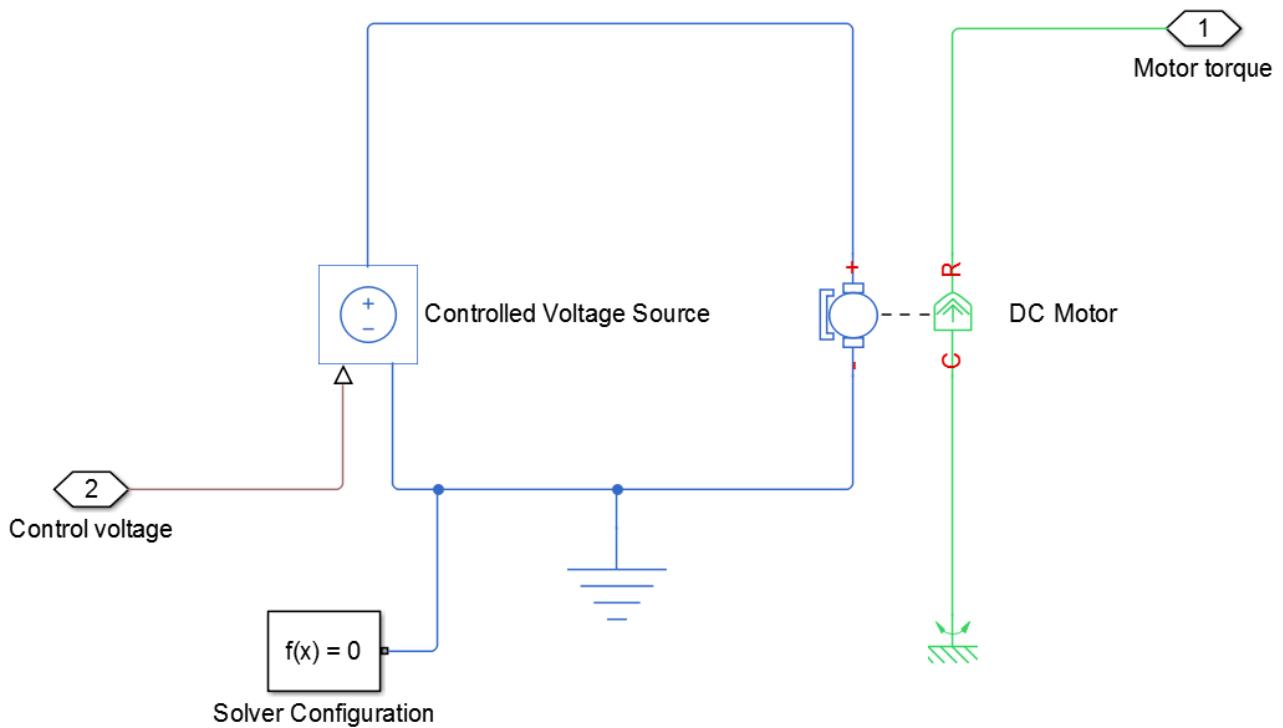


Figure 123: renaming the ports in a sub-system

Go back to the overall model view by clicking on the model name on the left side of the window. The Model Browser (situated on the left of the window) allows you to easily navigate from a sub-system to the global system (Figure 124).

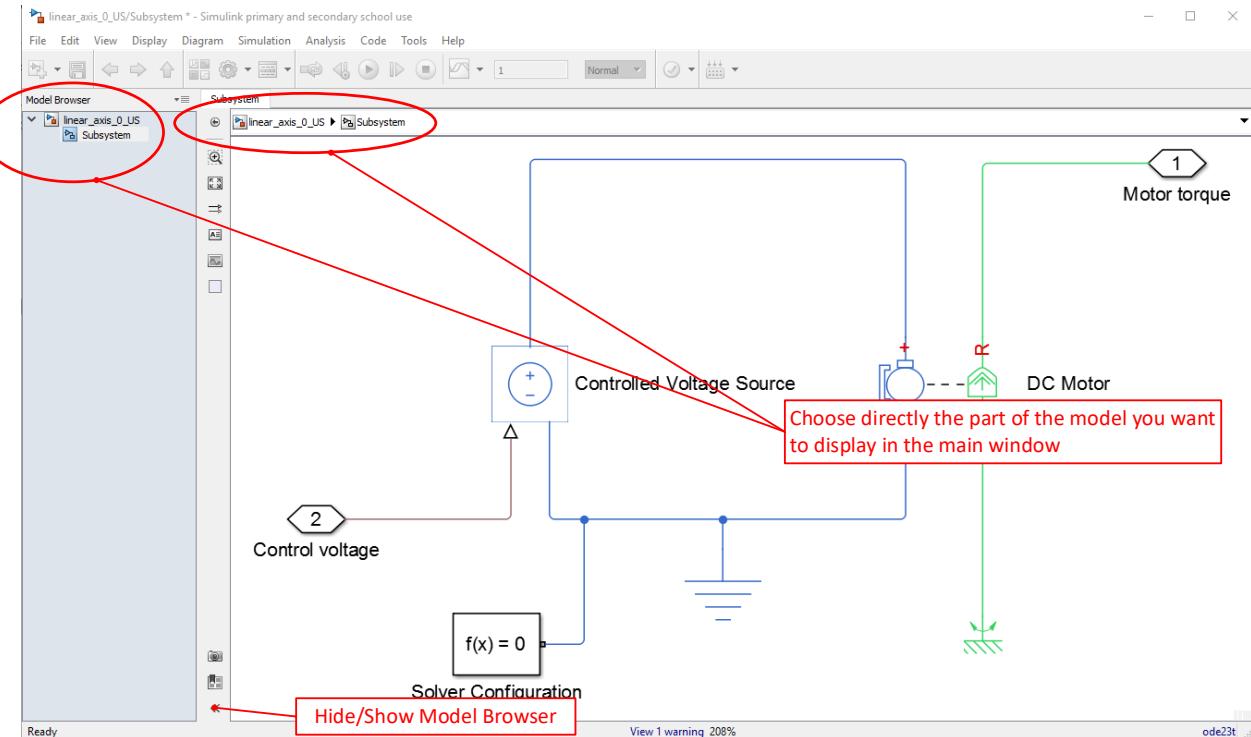


Figure 124: navigate in the sub-systems of a model

You need to obtain the configuration of **Erreur ! Source du renvoi introuvable..**

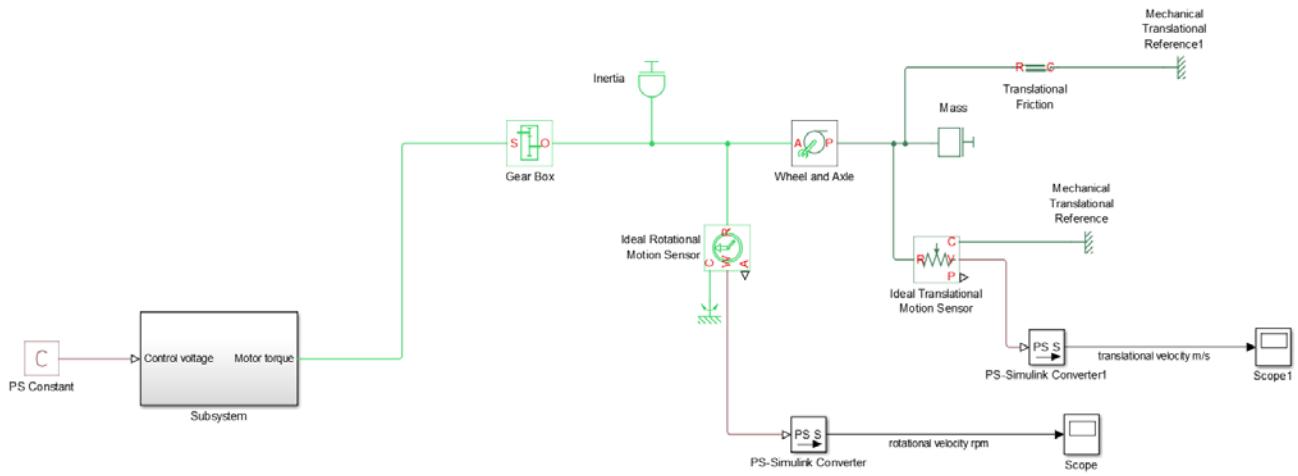


Figure 125: Simscape model of the linear axis with “motor” sub-system

With the objective of creating a position control, we will remove the rotational speed sensor of the motor shaft and transform the measurement of the linear velocity of the axis into a measurement of position.

Delete the rotational speed sensor in the motor shaft to get the diagram in Figure 126.

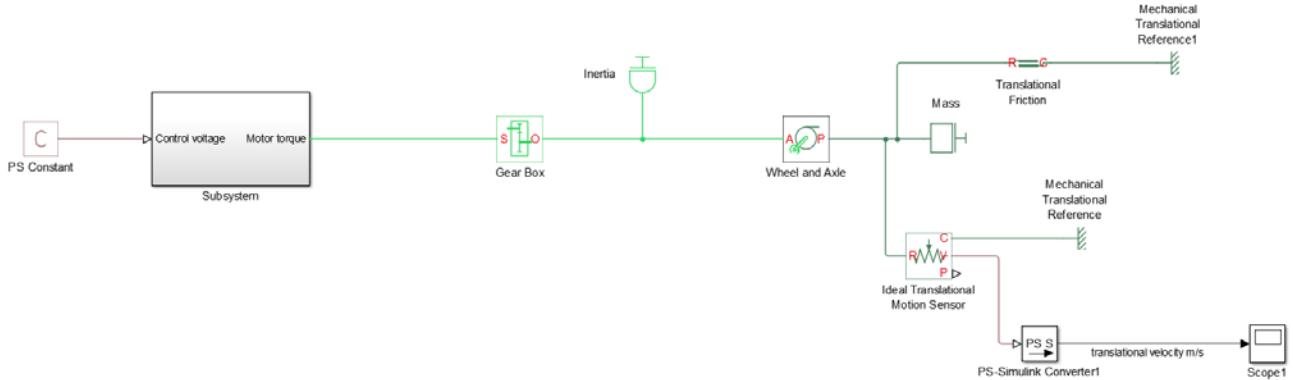


Figure 126: Simscape model of the linear axis without speed sensor

Modify the physical quantity taken from the **Ideal Translational Motion Sensor** to identify the linear position of the carriage(Figure 126).

You need to change the unit in the **PS-Simulink Converter** block and choose meter (**m**).

You also need to change the name of the signal to obtain the configuration of Figure 127.

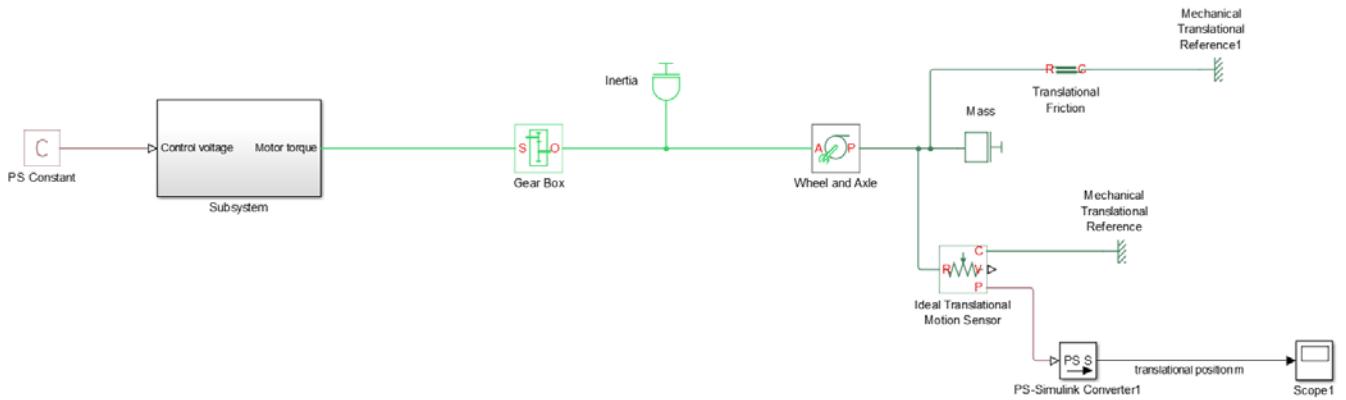


Figure 127: Simscape model of the linear axis with name of signals

Create an “Axis” sub-system by selecting the components in Figure 128.

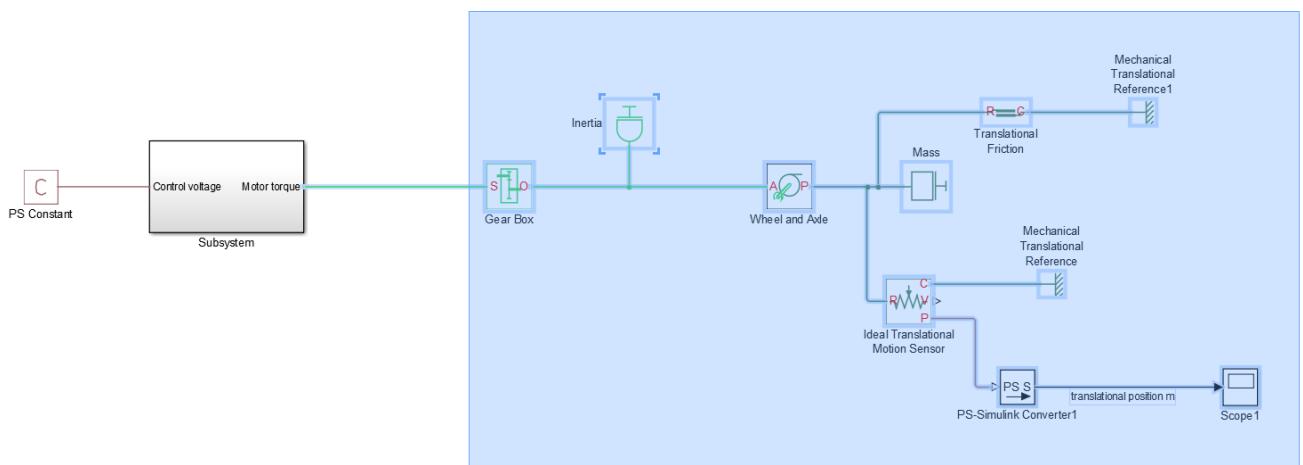


Figure 128: Simscape model of the linear axis, creation of the “Axis” sub-system

You should obtain the following model after resizing your sub-system:



Figure 129: Simscape model of the linear axis with “motor” and “Axe” sub-system

It can be noted that an additional port **Conn1** has been created.

Double-click on the sub-system to explore it.

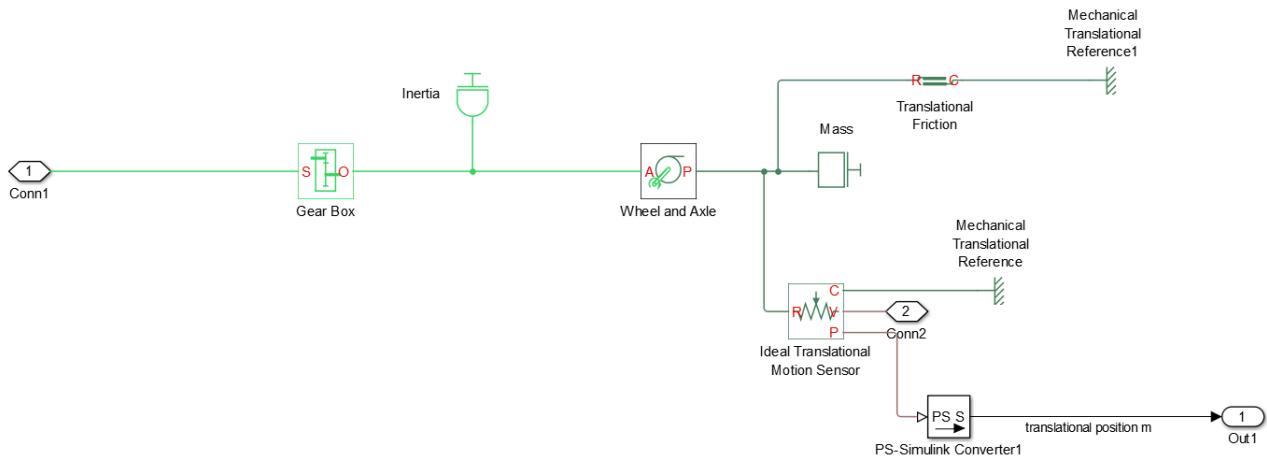


Figure 130: renaming the ports in a sub-system

The port **Conn2** had been created by default on the speed sensor exit which was not used. When creating a sub-system, any port which is not connected to the interior of the sub-system will appear as an exterior communication port.

Delete this port.

It should be noted that the format of port **Conn1** and that of port **Out1** are different. Port **Conn1** is a **PCP** type port in **Simscape** (transmitter of power, acausal modeling) and the port **Out1** is a **Simulink** signal (oriented digital signal, causal modeling). The software allows for the visual distinction between these two types of port.

Rename the ports to obtain the linear axis model in Figure 131.



Figure 131: finished Simscape model of the linear axis

If necessary, the corresponding model is available in the file **linear_axis_1_US.slx**

It is possible to add an image to a sub-system. Right click on the sub-system and select **Mask/Create Mask**

Type in the **Mask Editor** window (Figure 132) the command `image('axe.jpg')`. The corresponding image file must be in the **MATLAB** “path” to show up on the sub-system.

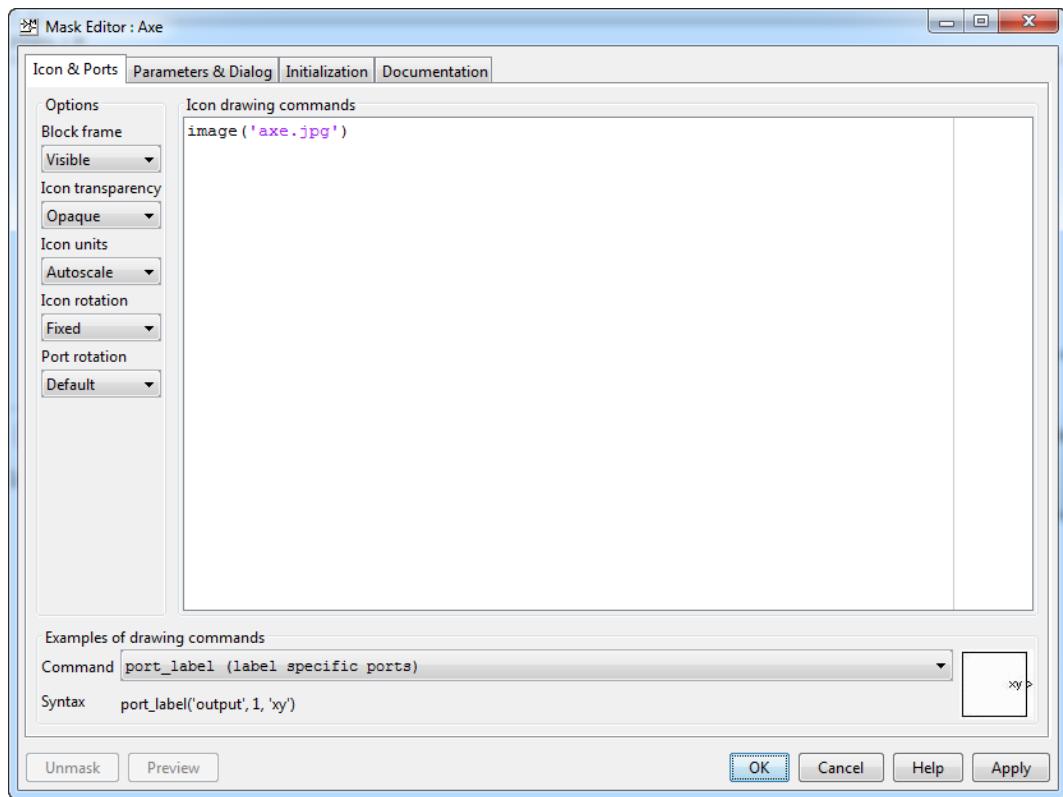


Figure 132: configuration of sub-system mask window

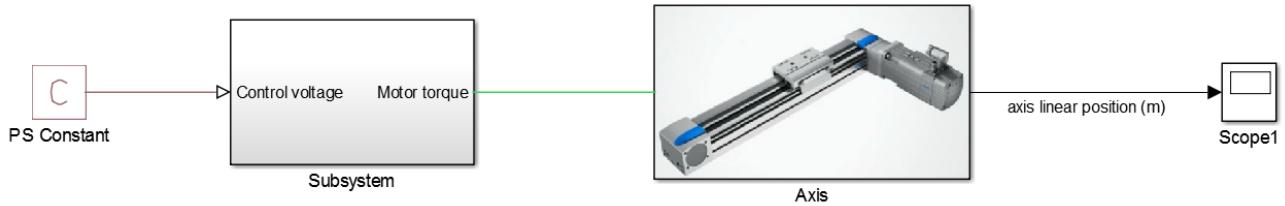


Figure 133: displaying an image on a sub-system

If necessary, resize the sub-system so that the image can be displayed with its original dimensions (Figure 133).

7. Modeling the servo position of the axis

The model thus obtained is used to view the open loop response of the position of the carriage for a given value of the motor voltage.

For a voltage of 12V, the response position of the axis is as follows:

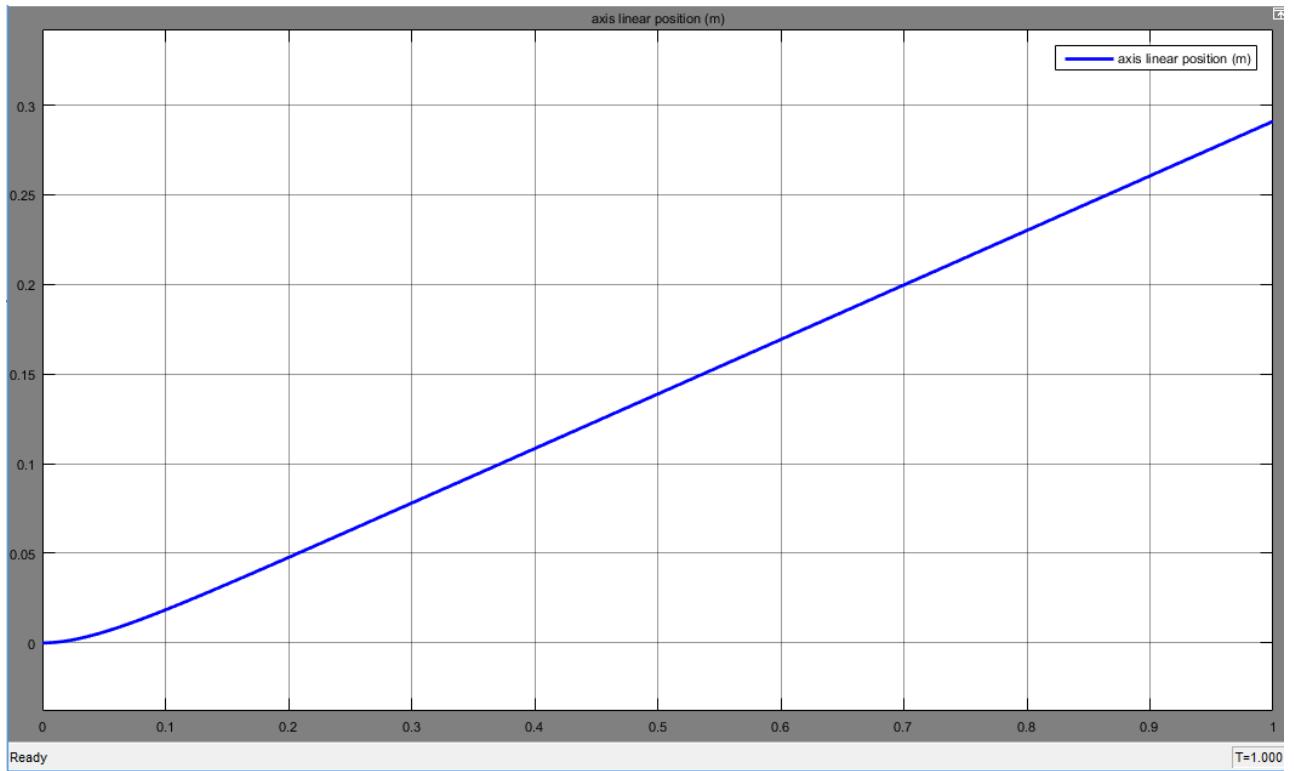


Figure 134: response position of free running linear axis

We can now create the servo control model with the axis position as in Figure 135. The newly added components are described in Figure 136.

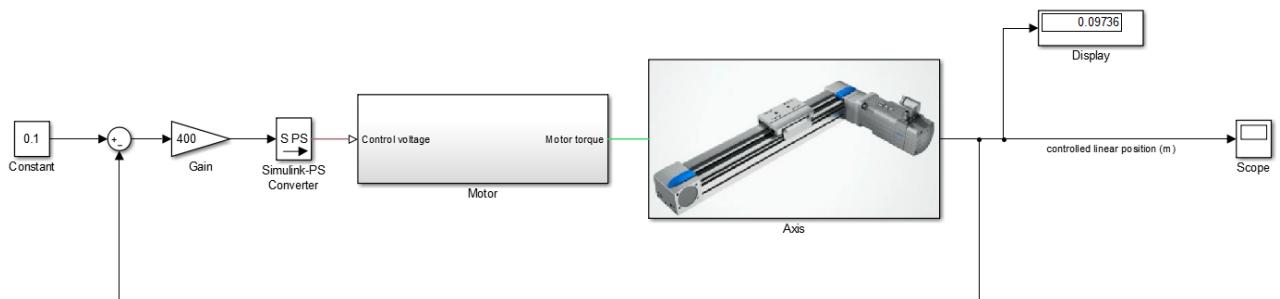


Figure 135: model of the servo controller in linear position on the axis

Apply a setpoint of 0.1m. This setpoint is compared to the measurement in the actual position of the axis. A proportional compensator transforms the difference in position in the setpoint for the motor voltage.

Only components that have not yet been used previously in this document are referenced in the table shown in Figure 136.

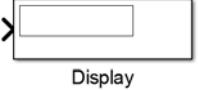
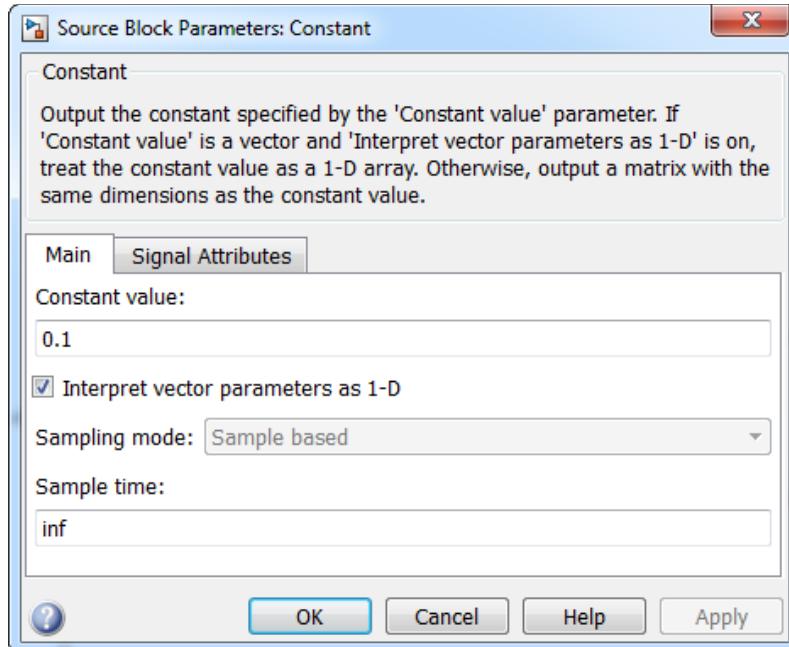
Component function	View	Library
Constant		Simulink/Sources or Simulink/Commonly used Blocks
Gain		Simulink/Math Operations or Simulink/Commonly used Blocks
Adder		Simulink/Math Operations or Simulink/Commonly used Blocks
Display		Simulink/Sinks

Figure 136: components to add to a model

Configuration

CONSTANT		Simulink/Sources or Simulink/Commonly used Blocks
----------	--	---

This block outputs a constant digital signal that corresponds here to the position setpoint of 0.1m.



Configuration

SUM



Simulink/Math Operations or Simulink/Commonly used Blocks

This component allows for a sum or a difference in two signals. The configuration corresponds to the succession of signs that are desired during the operation. Indicate in the **List of Signs** field available signs.

- $| - +$ for an adder of type



- $| - +$ for an adder of type

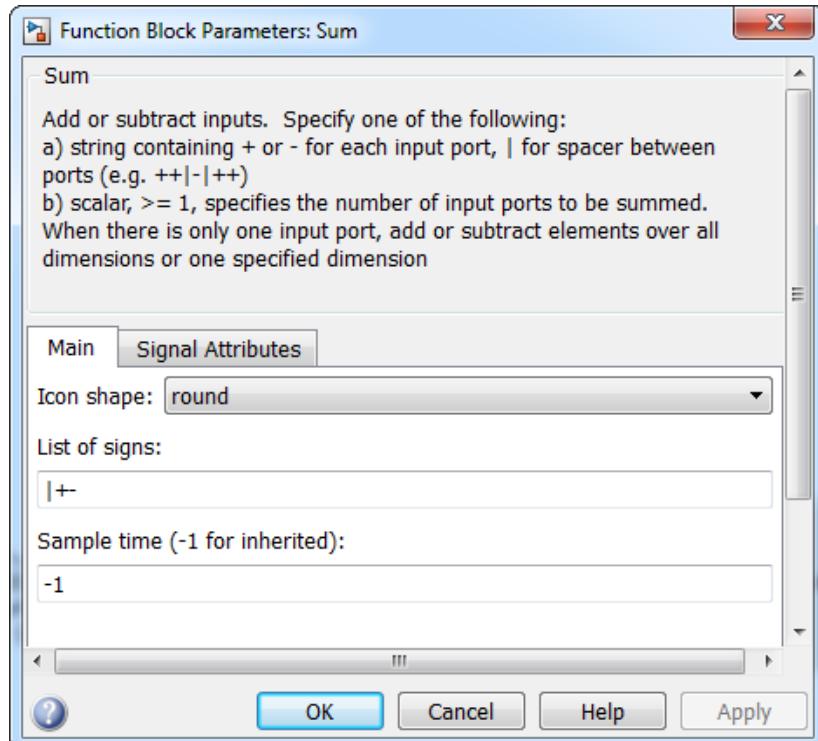
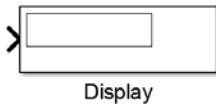


Figure 137: configuration of the adding block

Configuration

DISPLAY



Simulink/Sinks

This block enables a steady-state value to be displayed in order to view the precision of the position response.

Modify the template or open the file “**controlled_linear_axis.slx**”

Launch the simulation by specifying a simulation time of **1 second**.

View the response in the position using the scope.

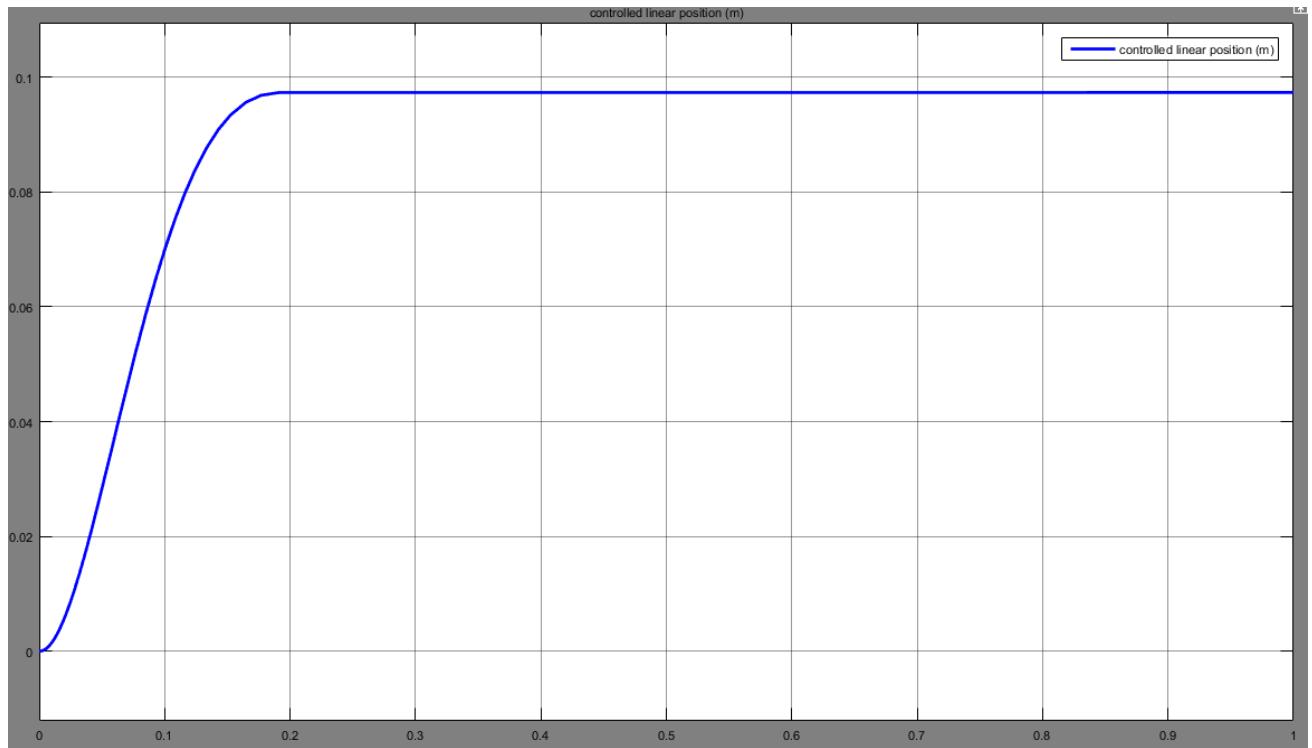


Figure 138: response of the dedicated axis in position

We note that the final position reached by the axis is 0.09736 m, which enables us to evaluate the static deviation at 3%. It is possible to improve the performance of the loop by using a PID controller.

Use the PID block to replace the gain block that represents the proportional regulator.

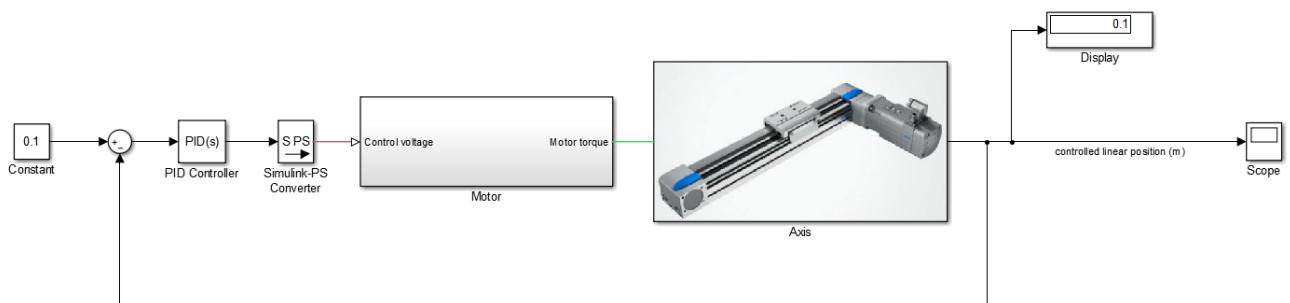


Figure 139: template for loop in linear axis position

Configuration



This block enables the PID controller to be configured. Here we will limit ourselves to inputting a proportional gain of 400 and an integral gain of 140 in order to cancel out the static error. We will return to the use of this block which possesses numerous interesting functionalities for the control command of systems.

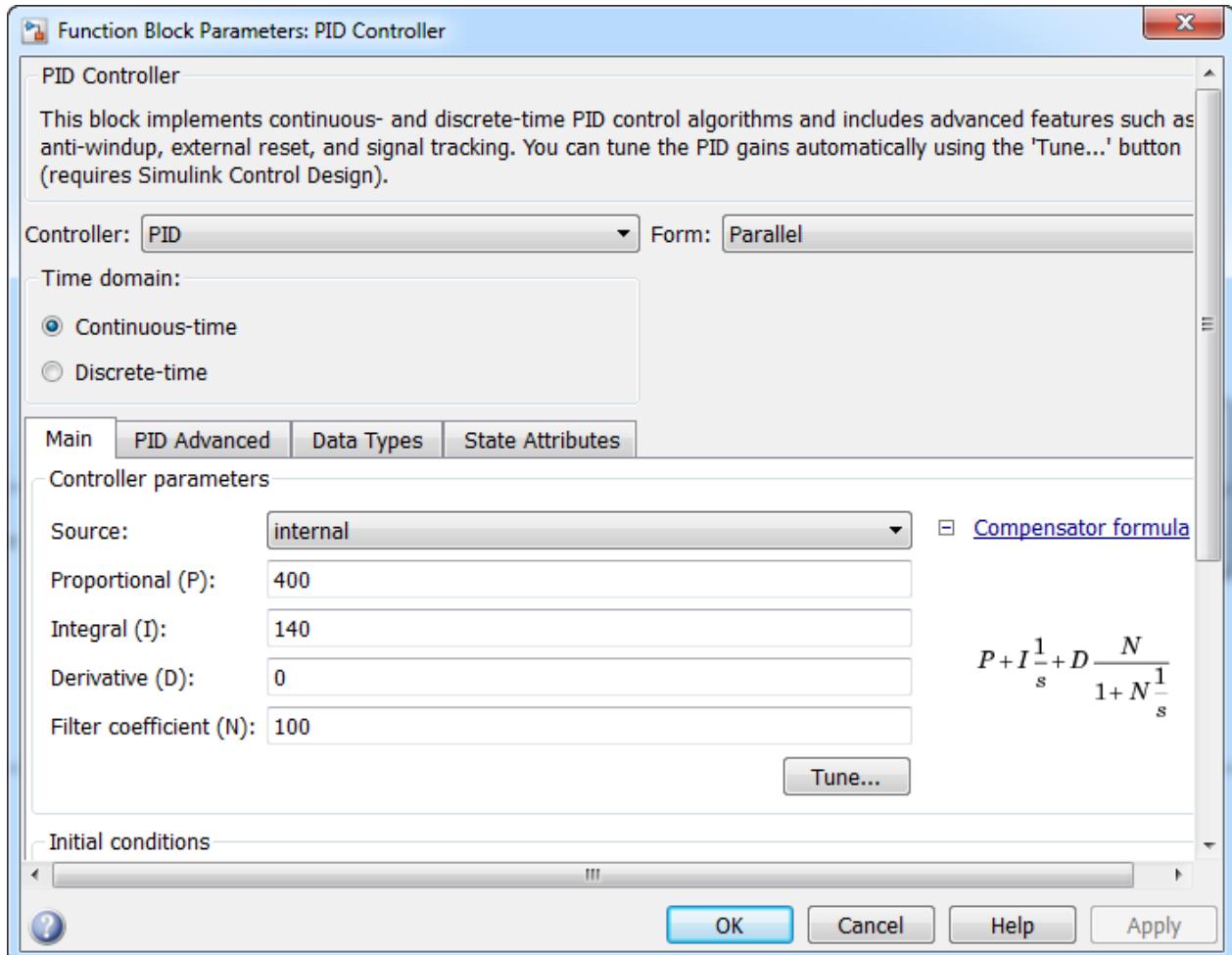


Figure 140: configuration of the PID block

Launch the simulation.

View the response in the position with proportional and integral correction.

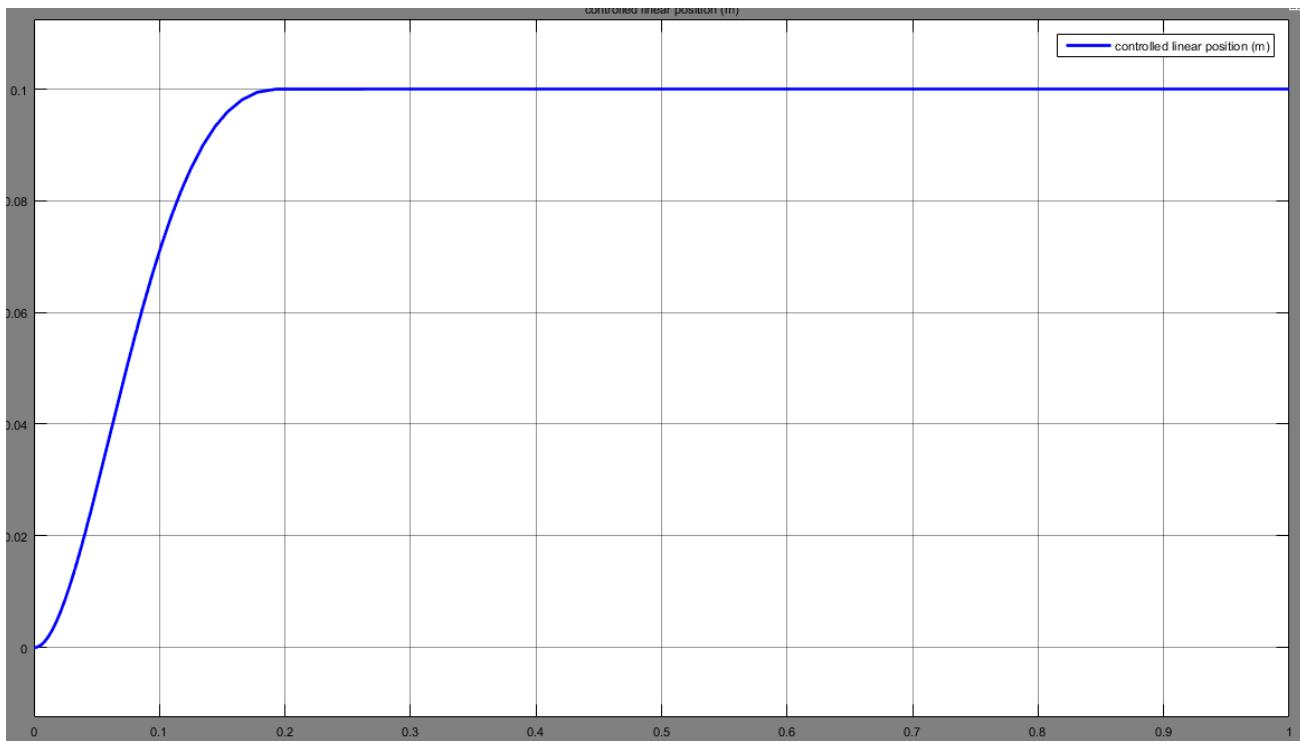


Figure 141: corrected response of the loop in linear axis position

We see the cancellation of the static error due to the presence of a time limit integrator in the PID.

The disturbance caused by the dry friction is compensated for by the PID controller for a value of 0.1 m.

B. Hydraulic-mechanical fields - single-acting hydraulic cylinder

The modeled system is a single-acting proportional hydraulic cylinder controlled by a 3/2 proportional valve, supplied with constant voltage. The displacement of the shaft causes the displacement of a weight that acts on the entire spring+shock absorber assembly. The objective is to estimate the progress of the speed and the position of the shaft as a function of time.

The system studied involves 2 physical fields:

- Hydraulic field
- Mechanical field

1. Choice of components

Only components that have not yet been used previously in this document are referenced in the table shown in Figure 142.

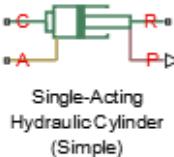
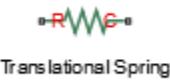
Component function	View	Library
Configurable signal source		Simulink/Sources
Hydraulic pressure source		Simscape/Foundation Library/Hydraulic/Sources
Proportional 3/2 distributor		Simscape/SimHydraulics/Valves/Directional Valves
Hydraulic reference		Simscape/Foundation Library/Hydraulic/Hydraulic elements
Fluid properties		Simscape/SimHydraulics/Hydraulic Utilities
Simple effect hydraulic cylinder		Simscape/SimHydraulics/Hydraulic Cylinders
Spring in translation		Simscape/Foundation Library/Mechanical/Translational Elements
Shock absorber in translation		Simscape/Foundation Library/Mechanical/Translational Elements

Figure 142: the components required for modeling the command from a hydraulic cylinder

The Simscape model of the single-acting hydraulic cylinder is represented in Figure 143 and enables the various physical fields that are involved in the model to be visualized. The interaction between the connections of the physical field and the actual components is provided.

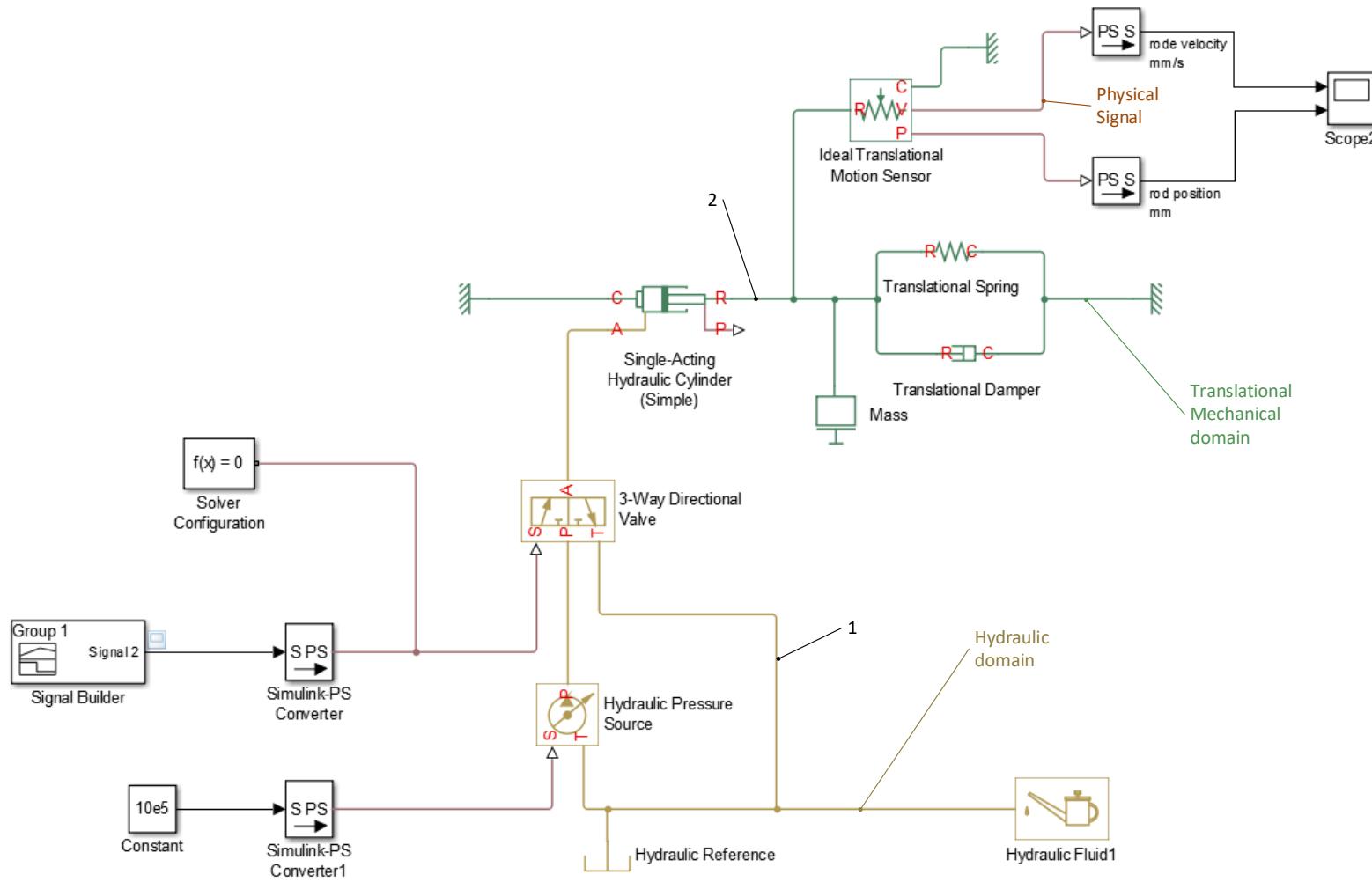


Figure 143: view of the physical fields involved in modeling the hydraulic cylinder control

1. Placement and assembly of components

In the **Simulink Library Browser** window, execute the **File/New/Model** command to create a new file.

Drag/Drop the various blocks from the libraries, place them in the working window, and link them to obtain the configuration shown in Figure 144.

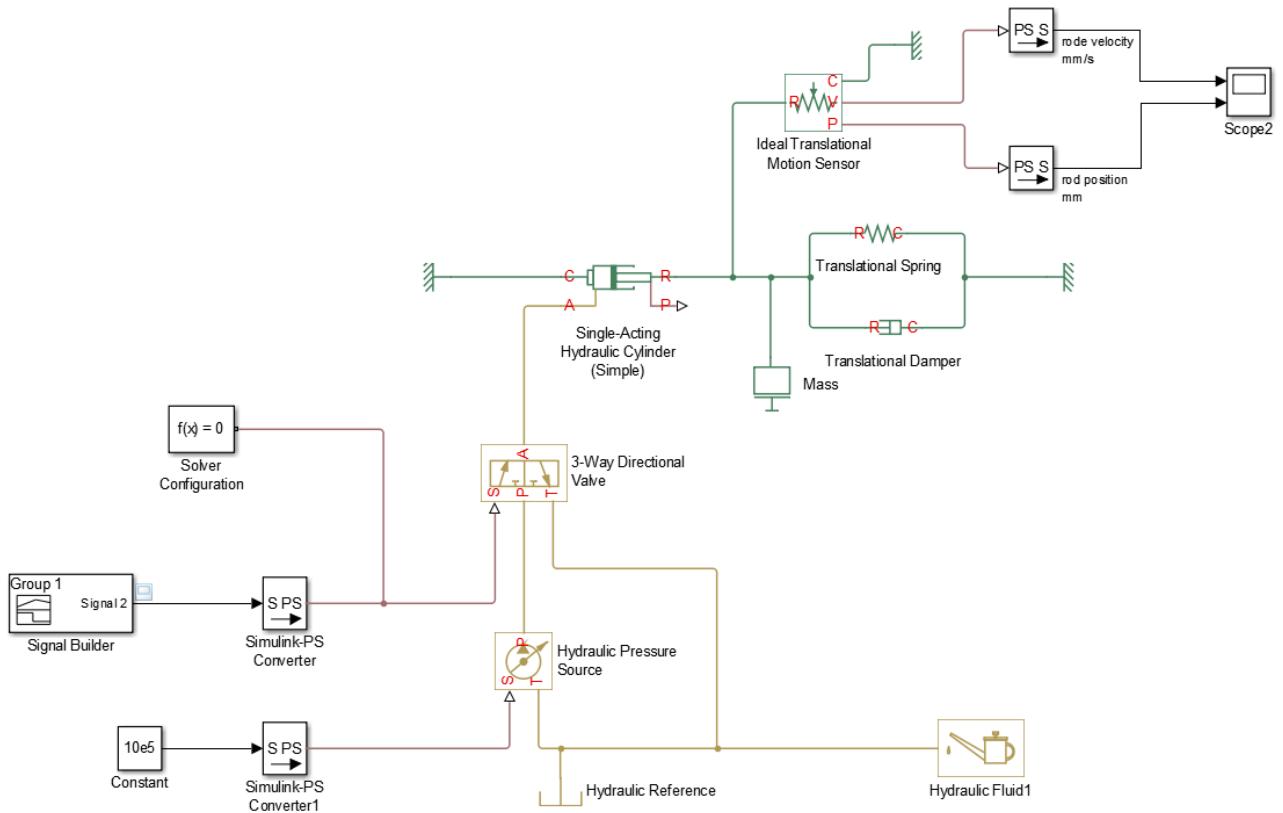


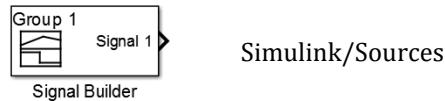
Figure 144: Simscape model of the hydraulic cylinder control

2. Configuration of components

Configure the various blocks in accordance with the information provided below.

Configuration

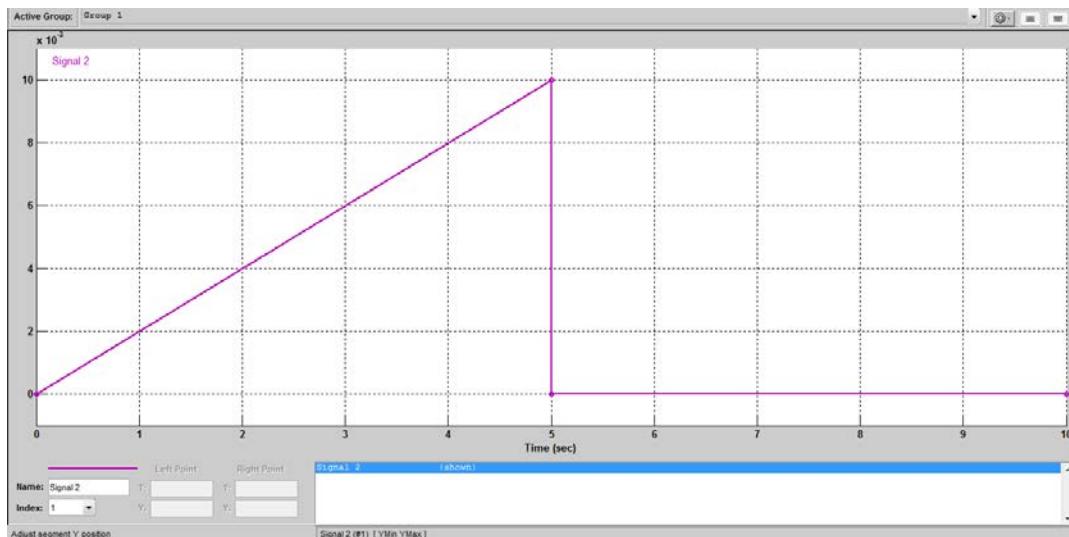
Signal Builder



Simulink/Sources

This block enables command signal of any type to be created. Here we would like to control the distributor in a linear manner from value 0 to value 0.01 corresponding to the maximum opening of the supply orifice of the actuator. From $t=5$ s, the command signal is brought to 0.

Refer to appendix “**Using the Signal Builder**” to configure the signal.



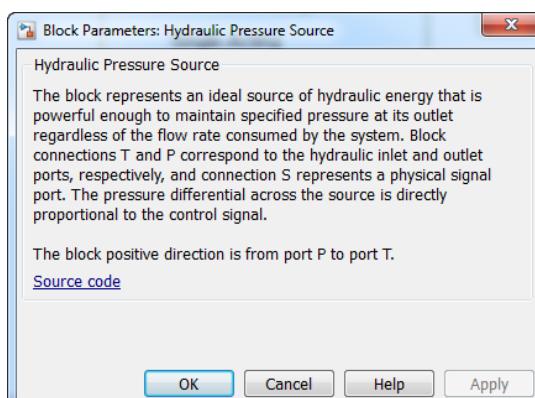
Configuration

Hydraulic Pressure Source



Simscape/Foundation Library/Hydraulic/Sources

This component enables a perfect pressure source to be modeled. The pressure will be constant regardless of the operation conditions. It possesses 2 PCP type ports (**T** and **P**) from the hydraulic field, and one physical signal type port that indicates the pressure value. Here the pressure will be regulated to 10^6 Pa. The “**Constant**” block supplies this pressure source through the intermediary of an **S-PS** block whose unit will be set to **Pa**.



Configuration

3 – Way Directional Valve

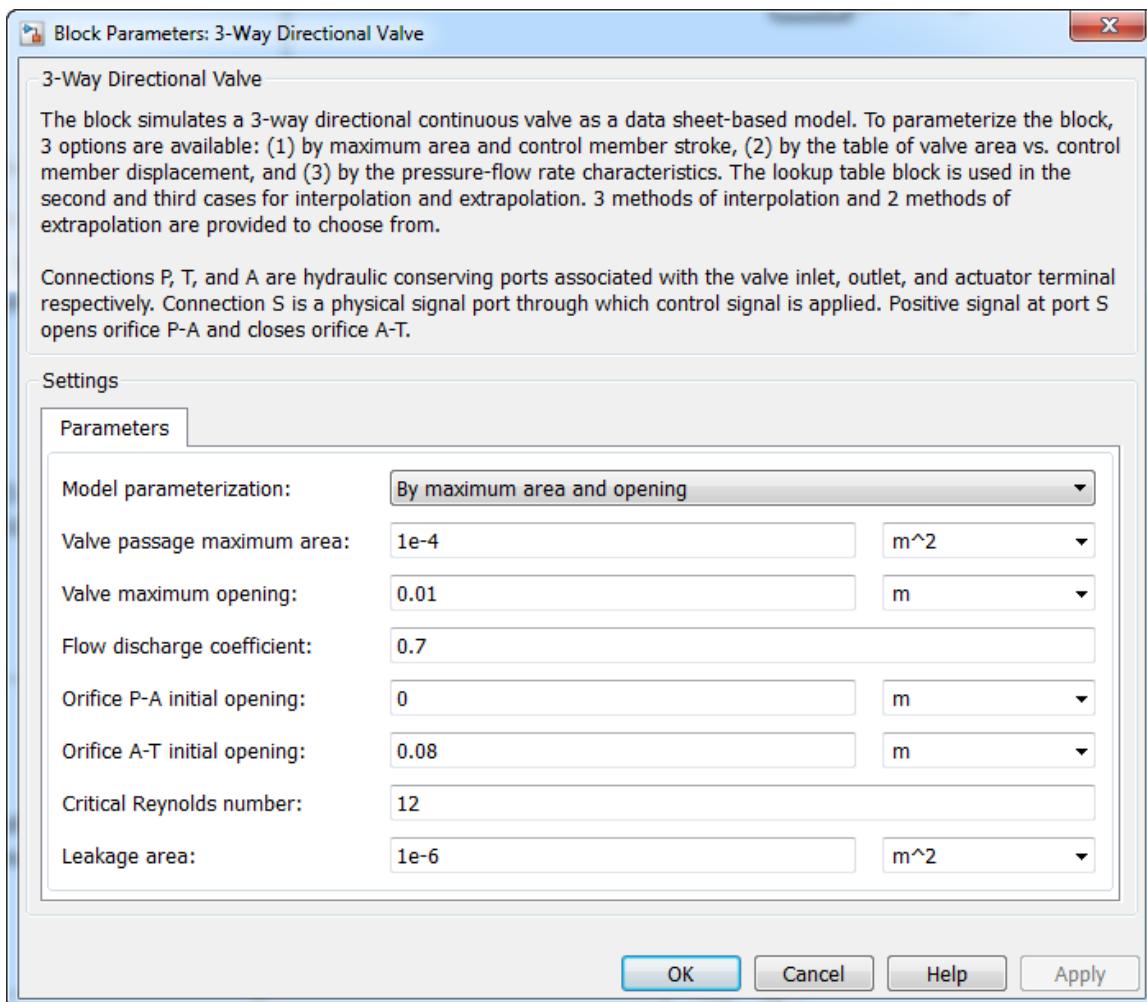


Simscape/SimHydraulics/Hydraulic Cylinders

This component enables a 3/2 hydraulic distributor with proportional control to be modeled. It possesses 3 **PCP** type (**T** and **P**) ports from the hydraulic field, and one physical signal type **S** port that indicates the pressure value.

- Port A: output to the actuator
- Port P: power supply
- Port T: emission

The configuration can be done in various ways based on the available manufacturer data (**Configuration Template** tab). Here the configuration is completed using **By maximum area and opening**.



Valve passage maximum area: indicates the maximum section for the passage of fluid

Valve maximum opening: indicates the value of the control that arrives at port S and gives the maximum opening of the valve.

Flow discharge coefficient: empirical coefficient that depends on the shapes of the servo-valve and what is provided in the manufacturer data (default value 0.7)

Orifice P-A initial opening (h_{PA0}): initial opening of the P-A orifice

Orifice P-A initial opening (h_{AT0}): initial opening of the A-T orifice

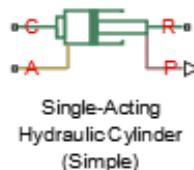
If we call x the command that arrives at port S, the respective openings of the orifices will be:

$$h_{PA} = h_{PA0} + x$$

$$h_{AT} = h_{AT0} - x$$

Configuration

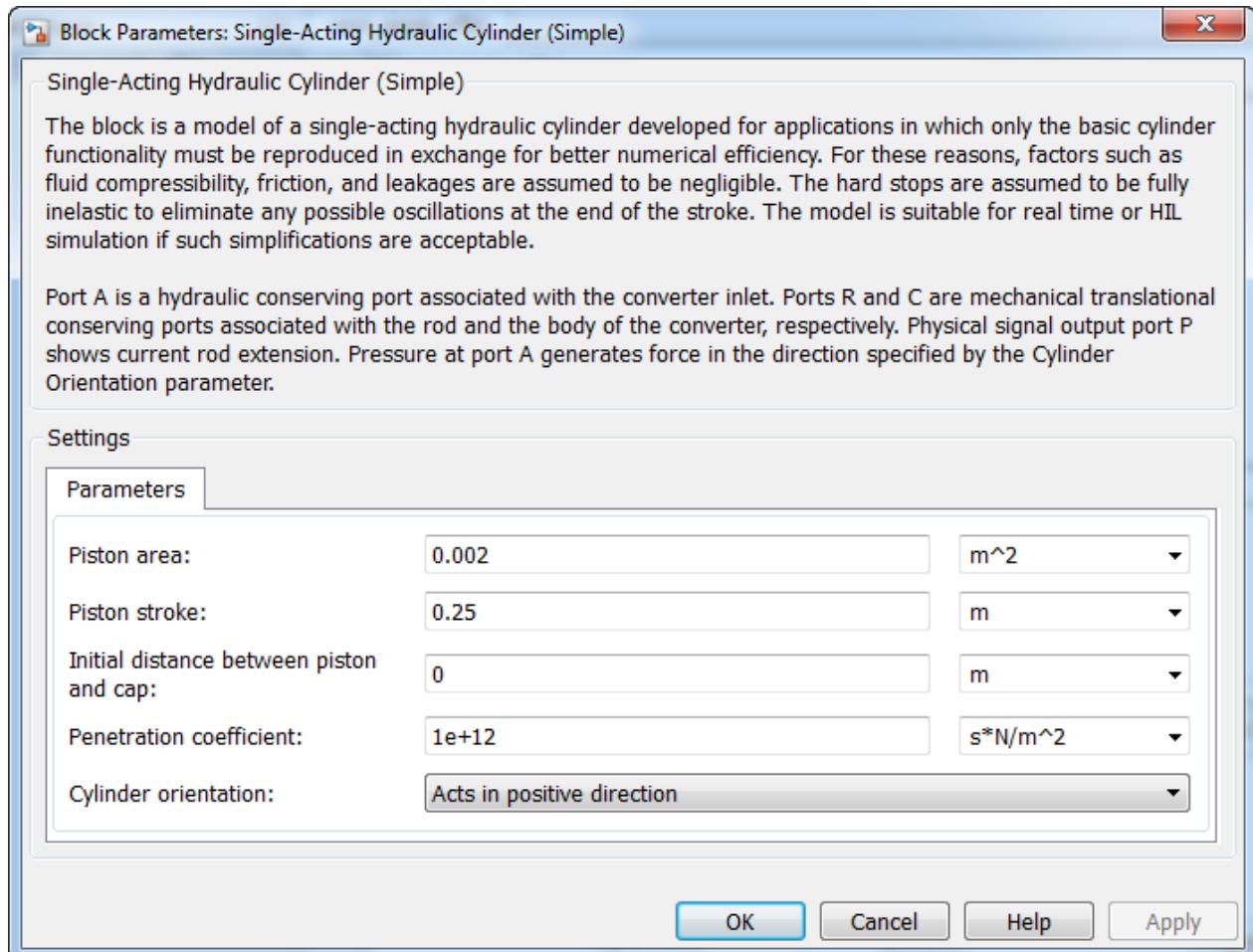
Single-Acting Hydraulic Cylinder (Simple)



Simscape/SimHydraulics/Valves/Directional Valves

This component enables modeling a single-acting hydraulic cylinder to be modeled. It possesses 3 PCP type ports, one from the hydraulic field (A) and two from the mechanical field of the translation (C and R)

- Port A: hydraulic cylinder supply orifice
- Port C: associated with the cylinder body
- Port R: associated with the cylinder shaft



Piston area: piston area

Piston stroke: path of the piston

Initial distance between piston and cap: initial position of the cylinder shaft (0 if the cylinder is fully returned at the start of the simulation).

Cylinder direction: enables the direction of the force generated at the end of the shaft to be chosen.

Configuration

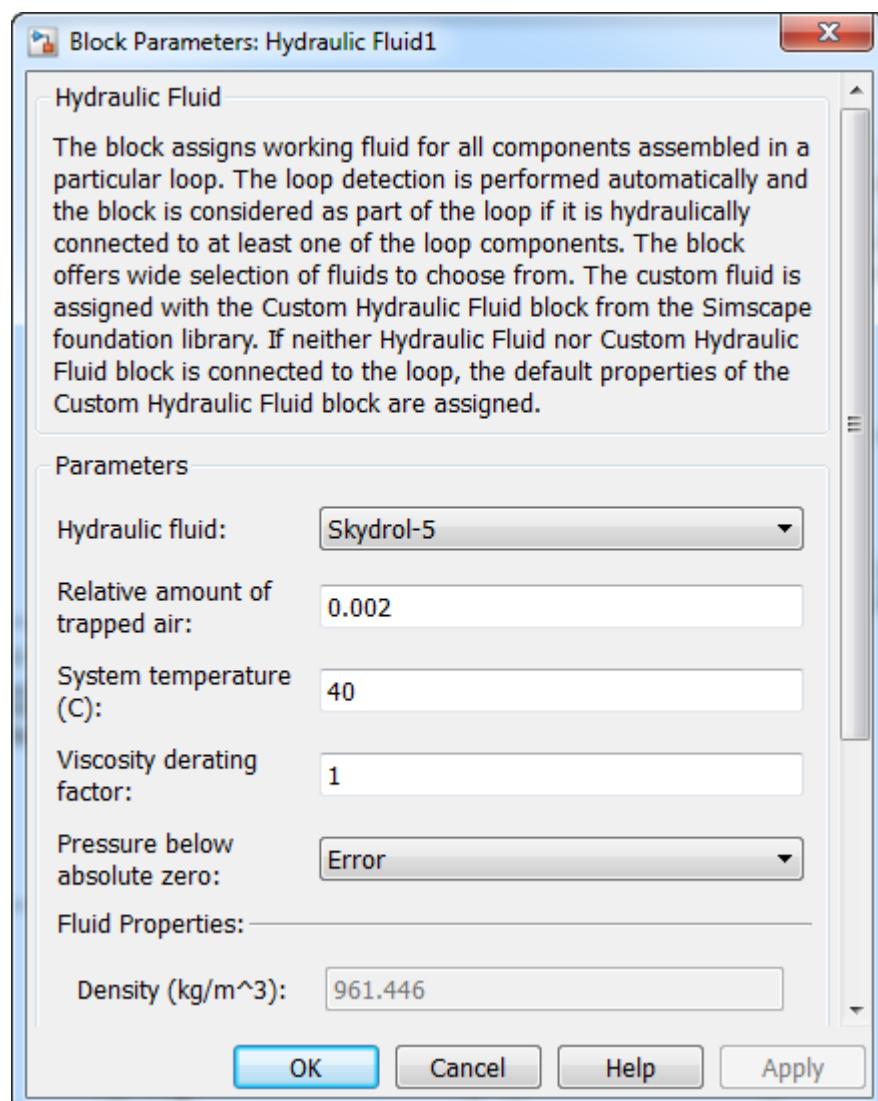
Hydraulic fluid



Simscape/SimHydraulics/Hydraulic Utilities

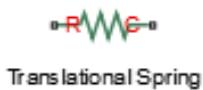
This component enables the properties of the fluid used to be modeled.

A block of the same type exists in the Simscape/Foundation Library/Hydraulics/Hydraulics Utilities library that enables the fluid characteristics to be entered manually if required.



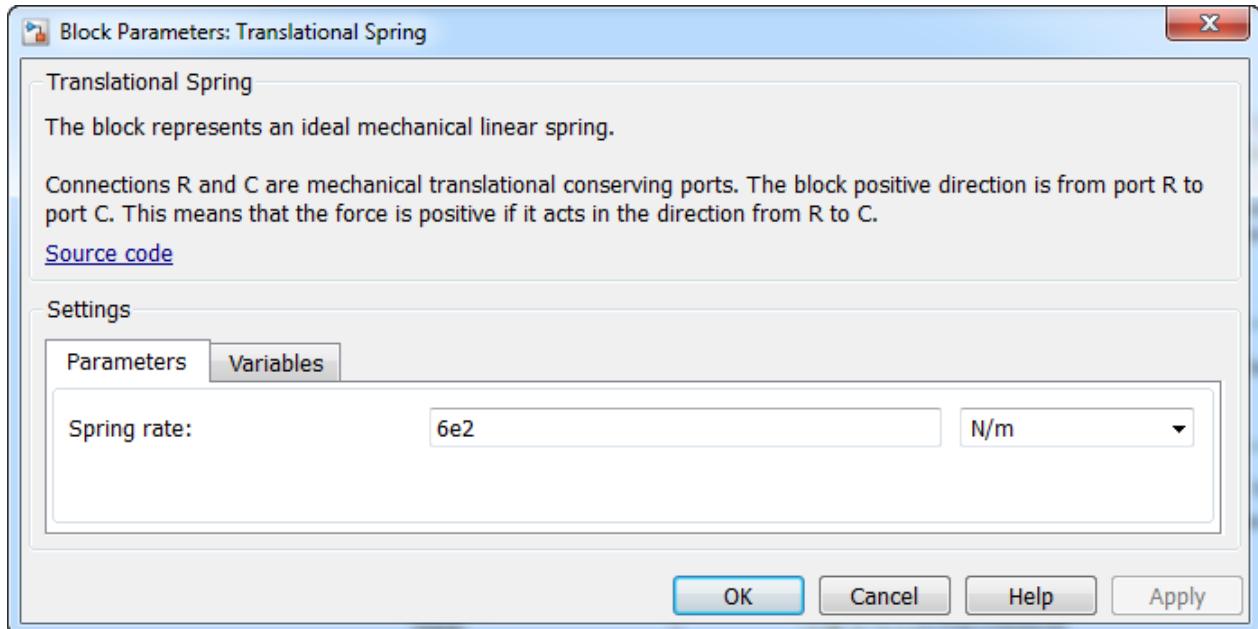
Configuration

Translational Spring



Simscape/Foundation Library/
Mechanical/Translational Elements

This component models a linear spring; the initial slope and length need to be specified.



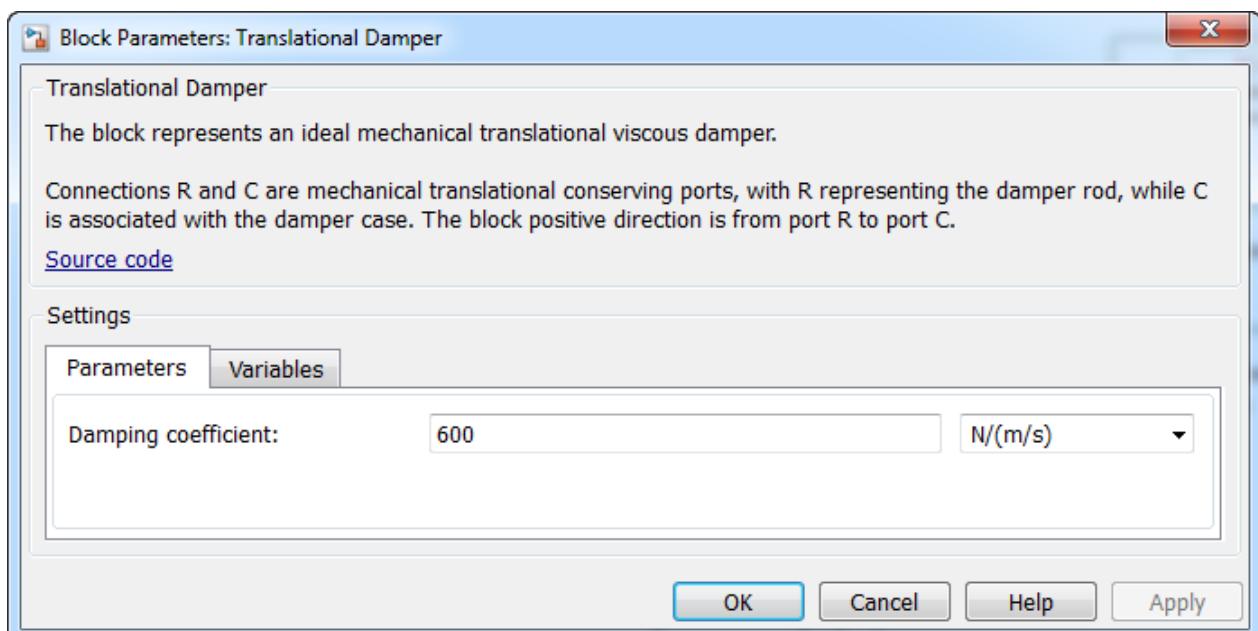
Configuration

Shock absorber in
translation



Simscape/Foundation Library/
Mechanical/Translational Elements

This component models a shock absorber; the shock absorbing coefficient needs to be modified.



Configuration

Scope



Simulink/Sinks

To configure a double-input scope, refer to the **Scopes** configuration appendix.

The file containing the configured model is available under the name **verin_simple_effet_pression.slx**.

3. Simulation

Choose solver **ode23t**.

Launch the simulation by specifying a simulation time of **10 seconds**.

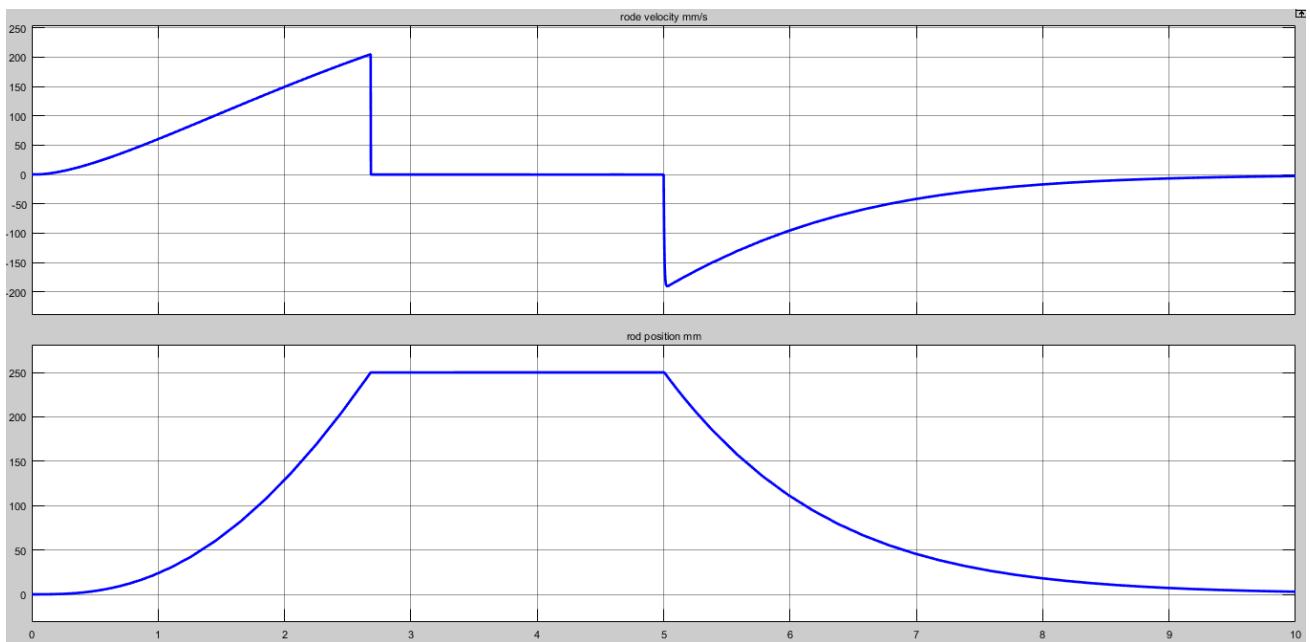


Figure 145: evolution of the speed and the position of the cylinder shaft

We observe in Figure 145 that the cylinder shaft exits, reaching the end of its path at $t=3\text{s}$. The shaft is stopped until the disappearance of the $t=5\text{ s}$ command. The shaft then returns to its initial position under the action of the spring.

4. Using the signal routing functionalities

It can sometimes be helpful to use the signal routing functionalities To simplify the results view.

Open the **single_acting_cylinder_tag_US.slx** file.

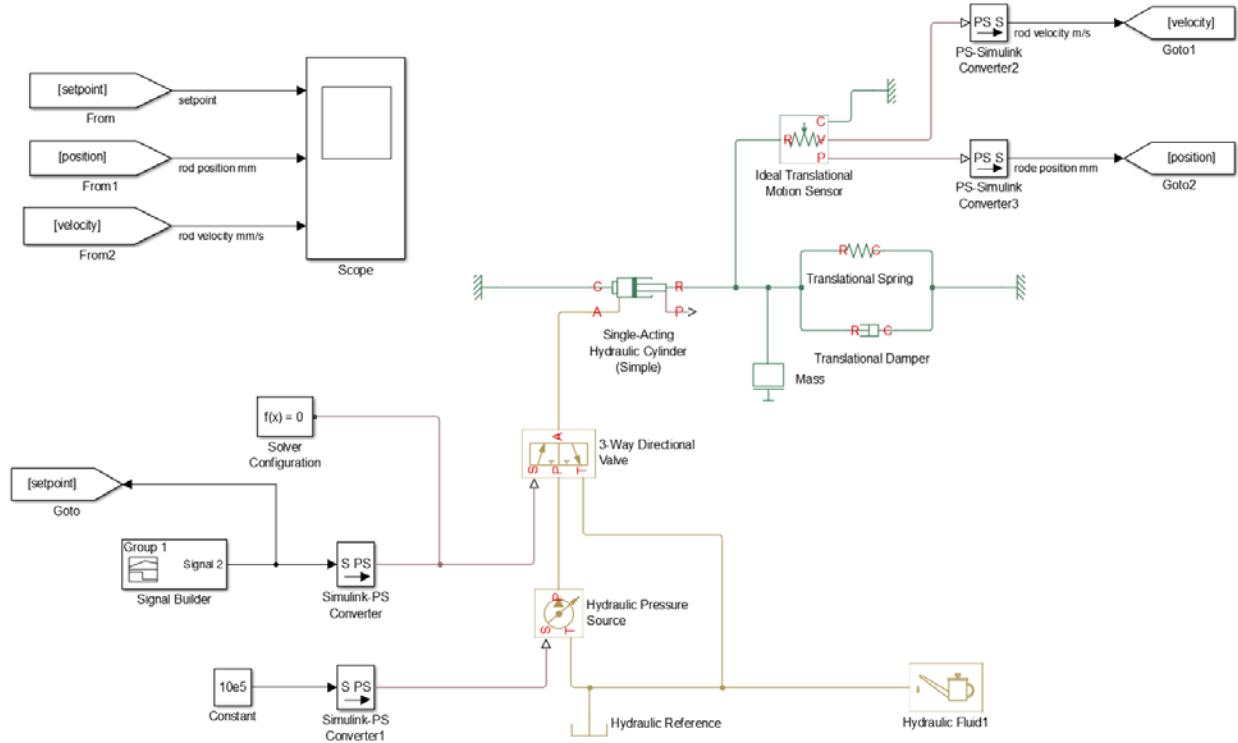
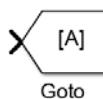


Figure 146: Simscape model of the hydraulic cylinder control with signals routing

This is a model of a single-action hydraulic cylinder control. The signals found on the model are shown here using a single scope. To avoid connections going through the model pointlessly, it is possible to route signals using the **Goto** and **From** blocks.

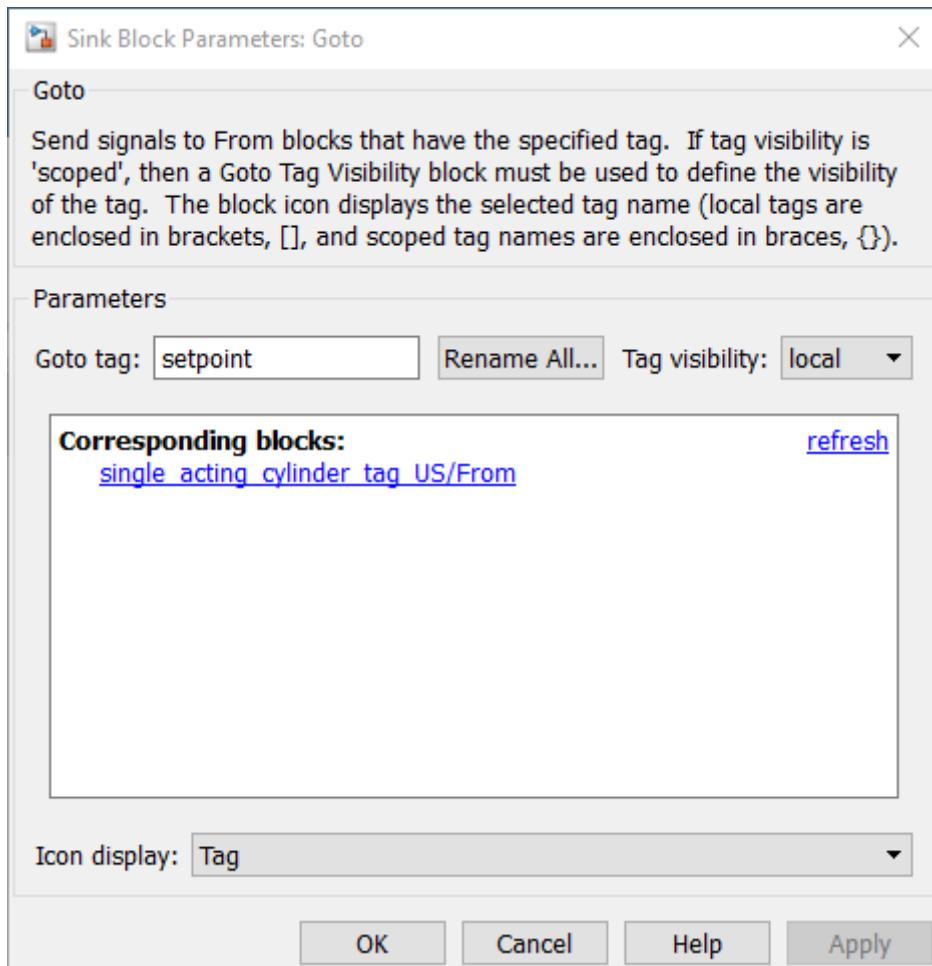
Configuration

Goto



Simulink/Signal Routing

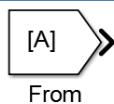
This block enables a **Tag** to be placed on a signal and the signal to be received using a **From** block.



Goto tag: indicates the name given to the signal

Tag visibility: indicates if the **Tag** is **Local** (only operable for the sub-system in which it is created) or **Global** (operable throughout the system)

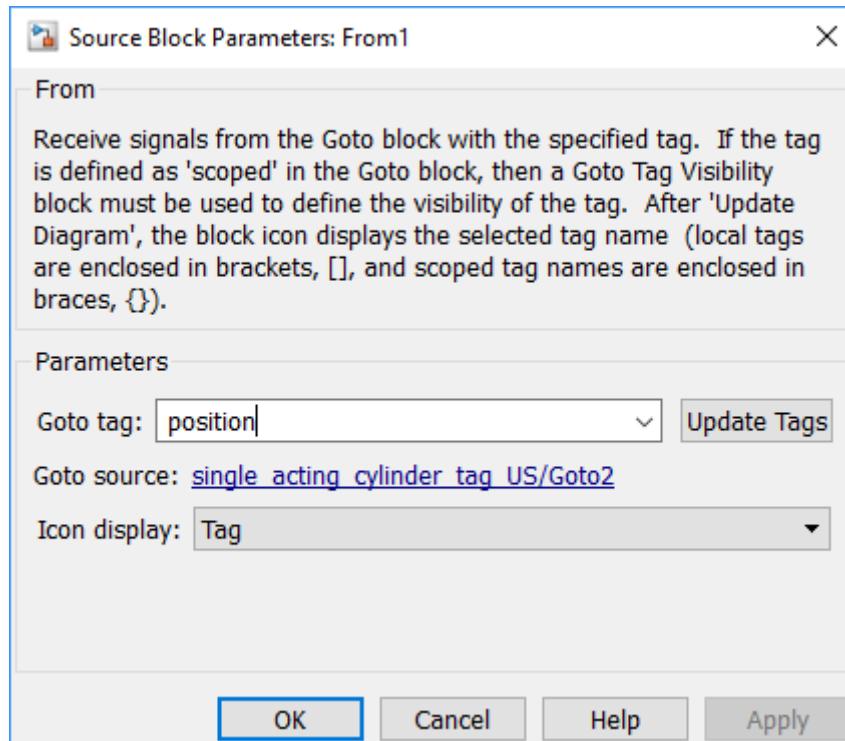
From



Simulink/Signal Routing

From

This block enables the Tag from a signal created using a **Goto** block to be received.



The Tag is selected directly from the dropdown menu. The **Update Tags** command refreshes Tags to make all Tags appear in the scrolling menu.

5. Replacing the pressure source with a flow source

The pressure source can be replaced with a flow source. In this case, a discharge valve will need to be provided so that the fluid can flow towards the reservoir when the cylinder shaft is stopped (Figure 147).

Open the **single_acting_cylinder_flow_source_US.slx** file.

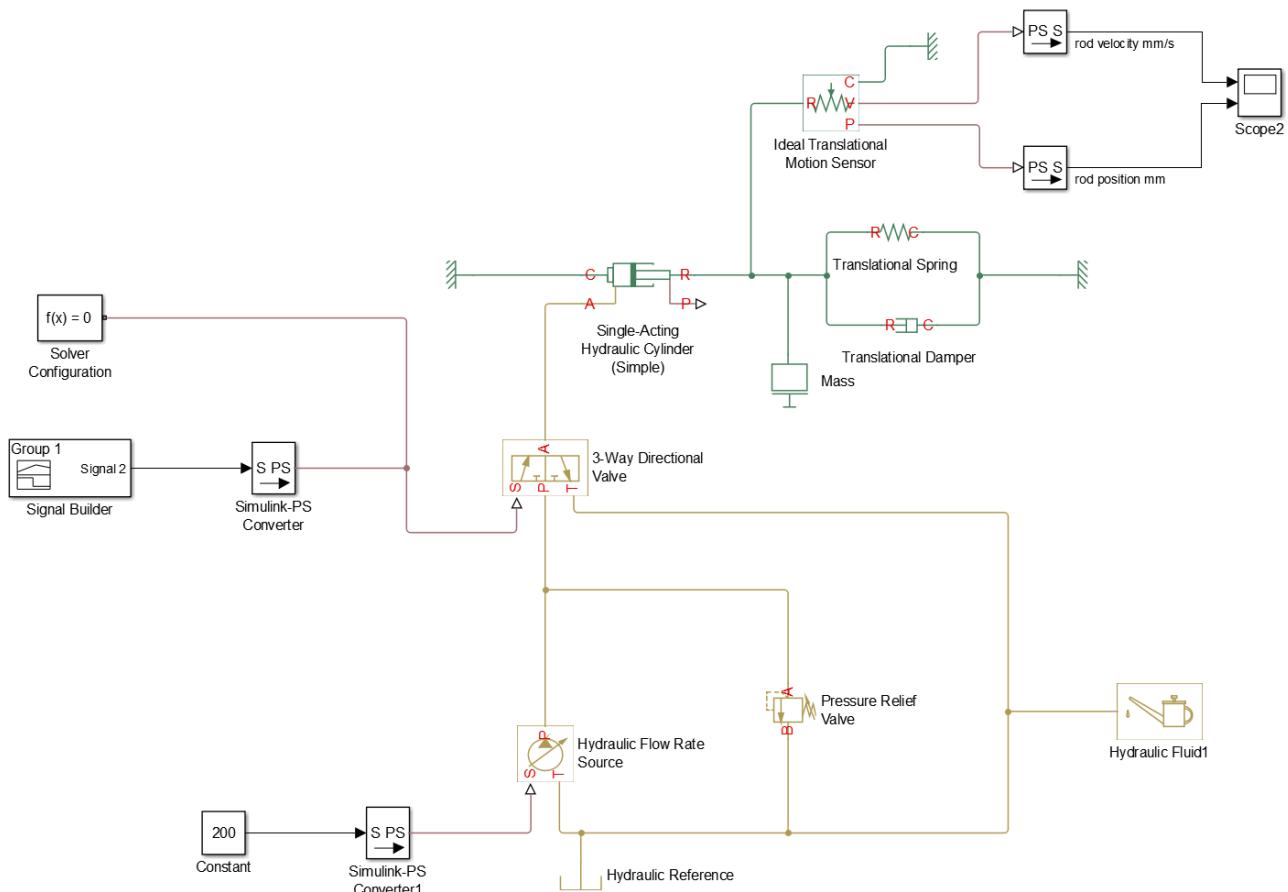


Figure 147: Simscape model of a hydraulic cylinder control with a flow source

The model is the same as previously. The pressure source has been replaced by a 200 L/min flow source. A discharge valve has been placed on the supply pipeline to limit the pressure when the shaft arrives at the stop position.

Configuration

Hydraulic Flow Rate Source



Hydraulic Flow Rate Source

Simscape/Foundation Library/ Hydraulic Sources

This a perfect flow source to be modeled. The flow will be constant regardless of the operation conditions. It possesses 2 PCP type ports (**T** and **P**) from the hydraulic field, and one physical signal type port that indicates the flow value. Here the flow will be regulated to 200 L/min. The “**Constant**” block supplies this flow source through the intermediary of an **S-PS** block whose unit will be set to **L/min**.

Configuration

Pressure Relief Valve



Simscape/SimHydraulics/Valves/Pressure Control Valves

This component models a pressure limiter.

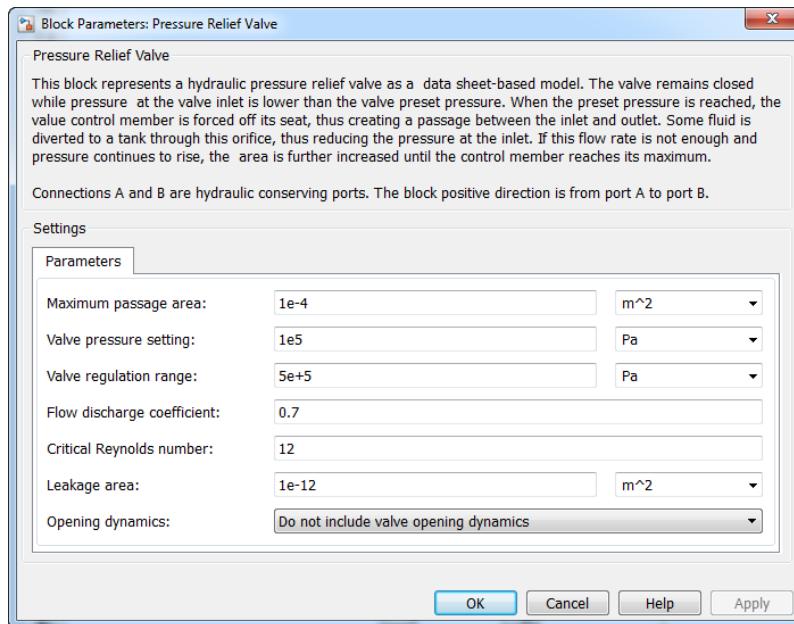


Figure 148: configuration of a pressure limiter block

Maximum passage area: indicates the maximum area for the passage of fluid

Valve pressure setting (p_{set}): indicates the pressure value at which the valve begins to open.

Valve regulation range (p_{reg}): indicates the range of pressure in the valve between the open and closed positions.

The curve in Figure 149 shows the variation in the opening of the valve depending on the pressure.

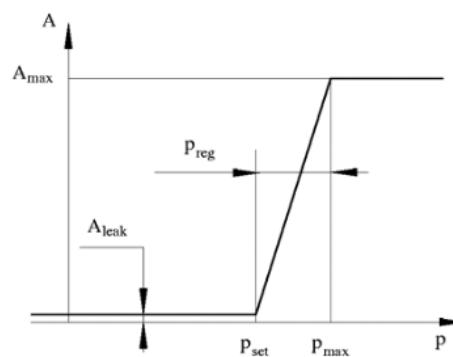


Figure 149: variation in the opening of the valve depending on the pressure

$$P_{max} = P_{set} + P_{reg}$$

Launch the simulation and observe the speed and the position of the cylinder shaft

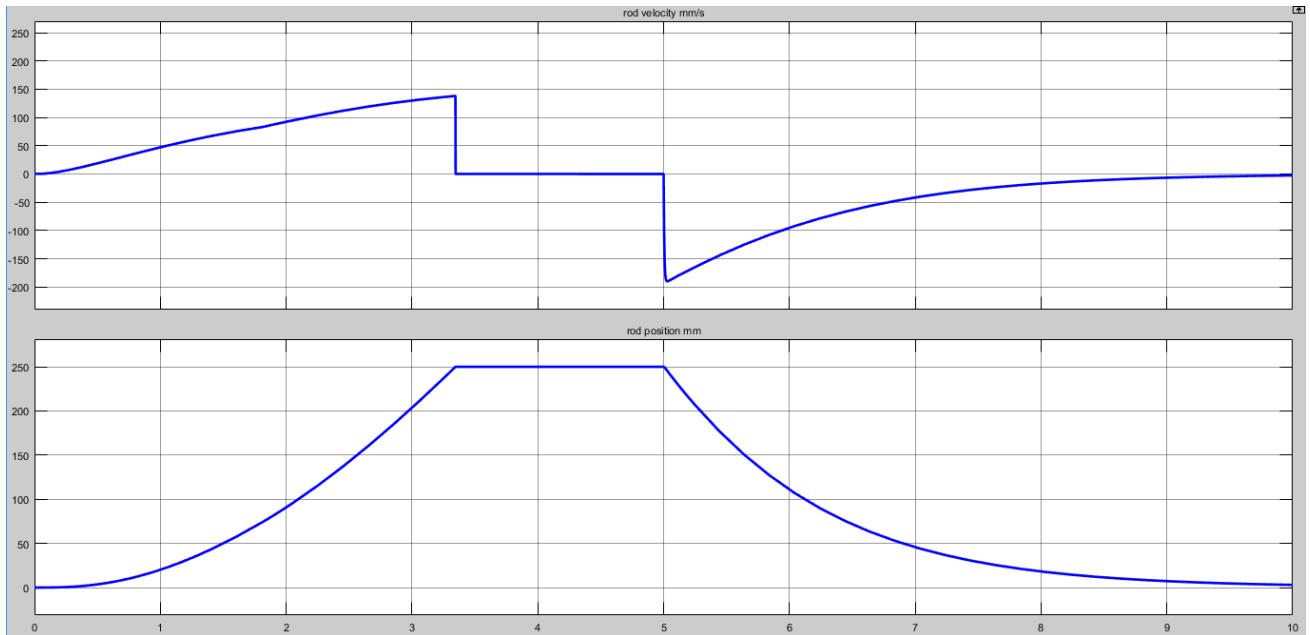


Figure 150: evolution of the speed and position of the cylinder shaft

We see that the shaft takes approximately 3.3 seconds to reach the stopper.

Increase the value of the **Valve pressure setting** in the pressure limiter to **1e6 Pa** and relaunch the simulation.

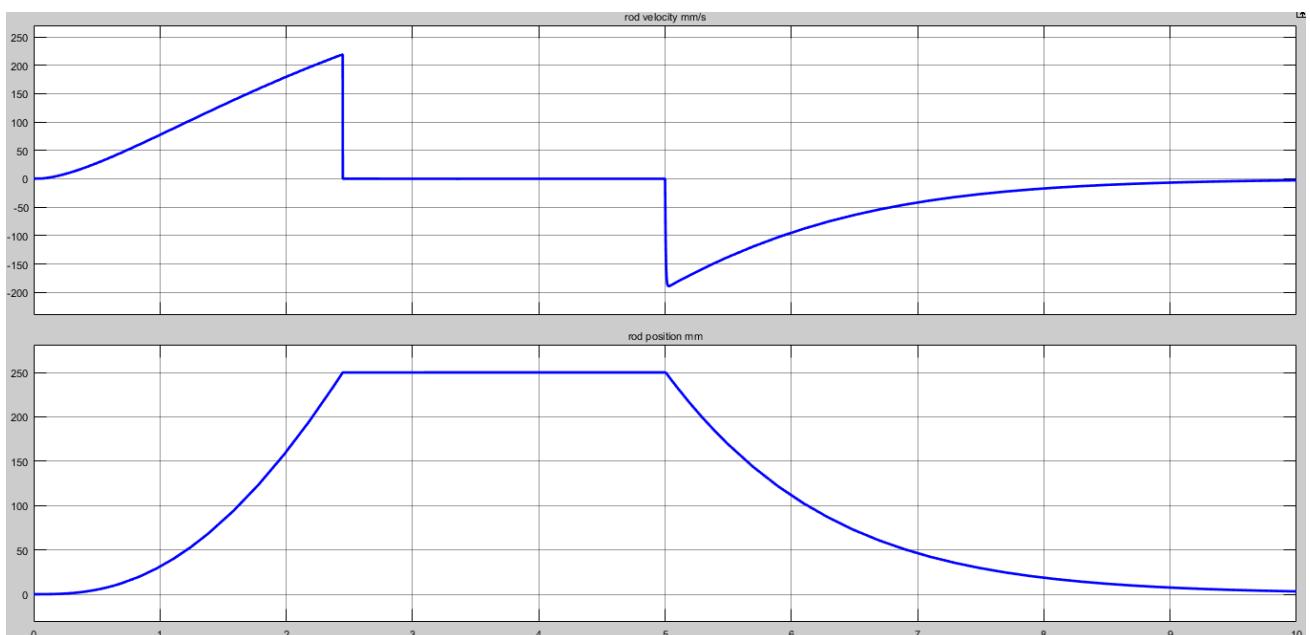


Figure 151: evolution of the speed and position of the cylinder shaft

We see that the increase in the admissible pressure in the cylinder's supply pipe enables the shaft to exit more quickly. The cylinder arrives at the stopper after 2.4 seconds.

C. Electric field - PWM control of a DC motor

In this section, we learn how to use the components required to design a PWM control for a DC motor.

A PWM control voltage is characterized by its frequency, its cyclic relationship (duty cycle) and its amplitude.

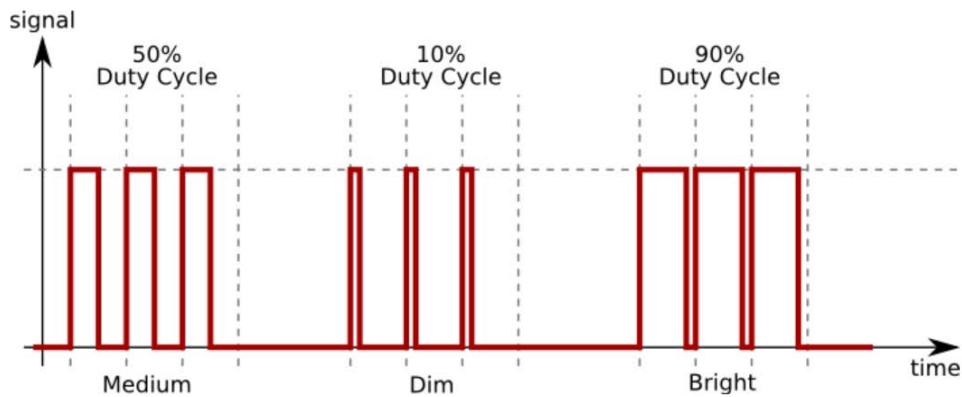


Figure 152: PWM control principle

To generate this voltage, we use the “**Controlled PWM Voltage Source**” component.

To reverse the direction of the motor or the stopper, we use the “**H-Bridge**” component.

These components are taken from the **SimElectronics** library:

Component function	View	Library
Generating PWM type control voltage		Simscape/SimElectronics/Actuators and Drivers/Drivers
H-Bridge		Simscape/SimElectronics/Actuators and Drivers/Drivers

1. Use of the “Controlled PWM Voltage” component

Open the **block_PWM_US.slx** file.

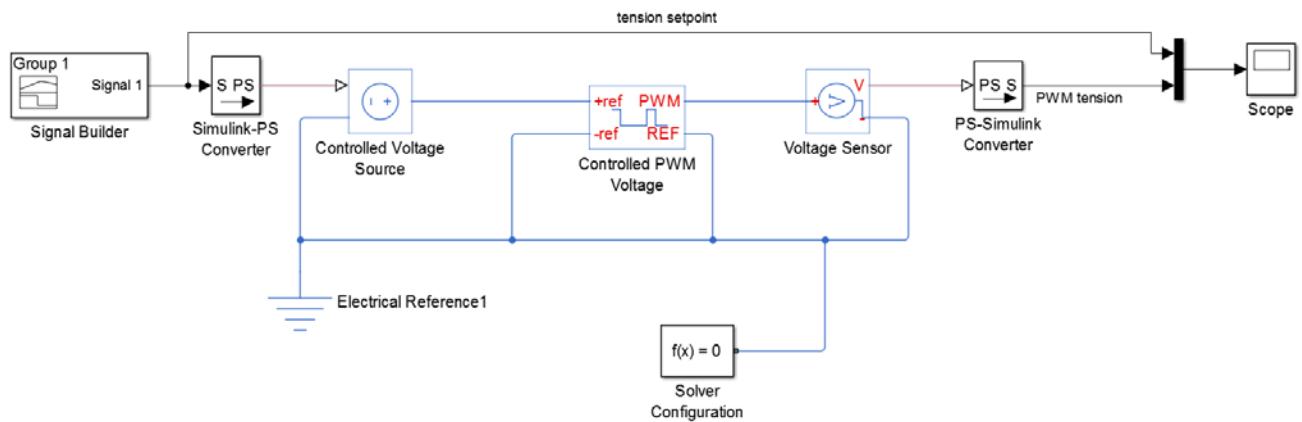


Figure 153: Illustration of how the “Controlled PWM Voltage” block functions

Double-click the **Signal Builder** to see the speed of the voltage placed on the bridge +ref (Figure 154).

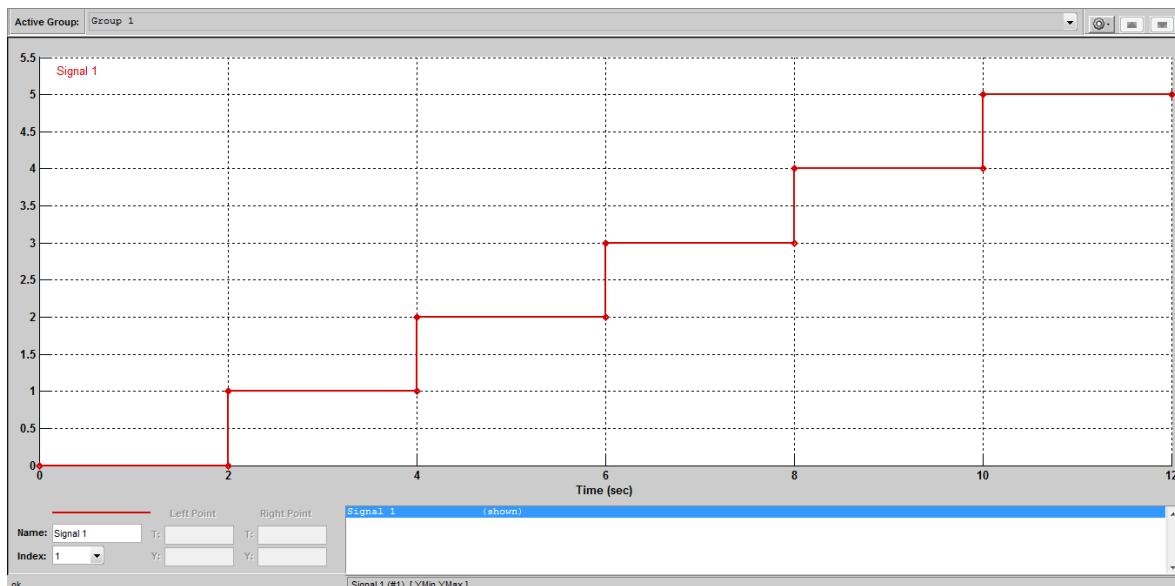


Figure 154: Controlled PWM Voltage block control voltage

This pressure varies from 0 V to 5 V and will cause the duty cycle of the PWM signal.

Launch the simulation and observe the result on the scope.

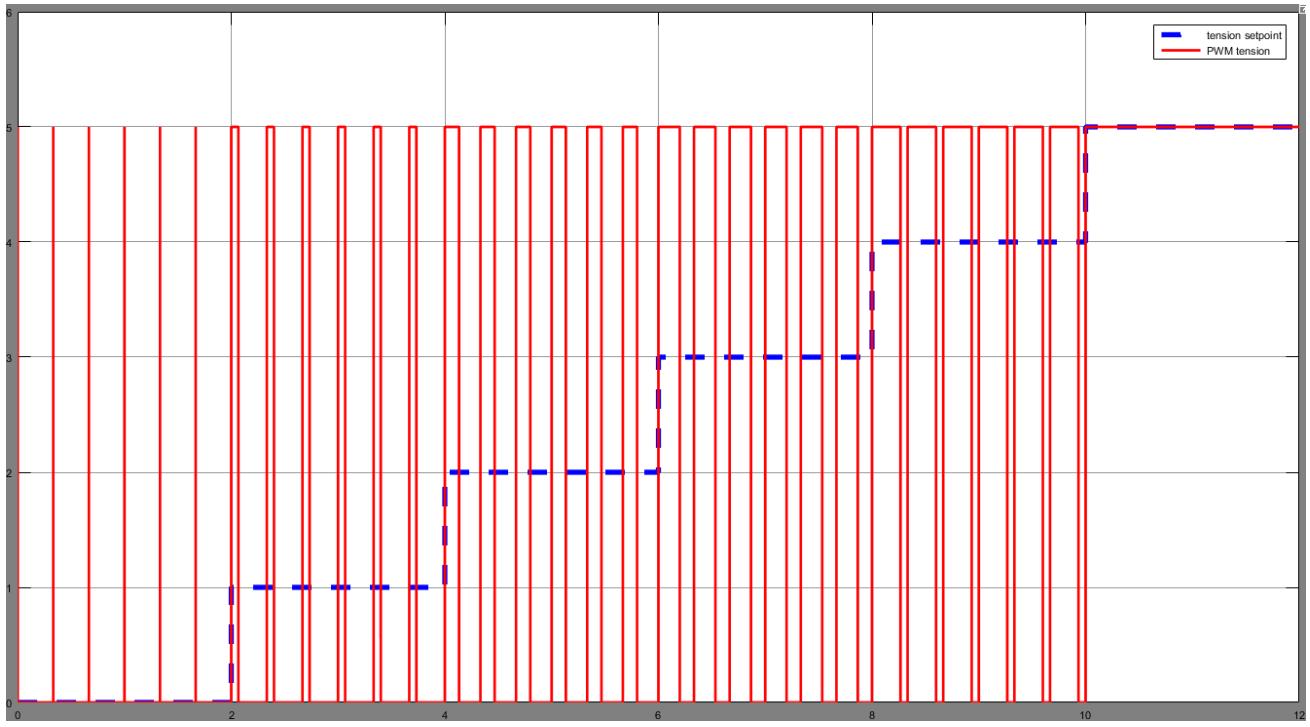


Figure 155: PWM signal based on the control voltage

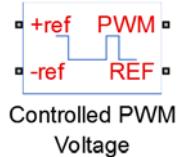
The control signal input on the port **+ref** is shown in blue; the PWM voltage exiting the **PWM** port is shown in red.

The frequency of the PWM signal depends on the amplitude of the voltage **+ref**.

The frequency and amplitude are introduced into the configuration of the component.

Configuration

Controlled PWM Voltage Source



Simscape/SimElectronics/Actuators and Drivers/Drivers

This component models a PWM voltage generator. It possesses 4 **PCP** type ports from the electric field. It enables a PWM pressure to be generated from the information taken from the input ports and from the component's configuration.

- **+ref**: this pressure will reflect the duty cycle of the PWM signal.
- **-ref**: pressure reference +ref (often associated with mass)
- **PWM**: PWM voltage signal for which the duty cycle depends on the entry +ref. The amplitude and frequency are specified in the component parameters.
- **REF**: voltage reference of the PWM signal (often associated with mass)

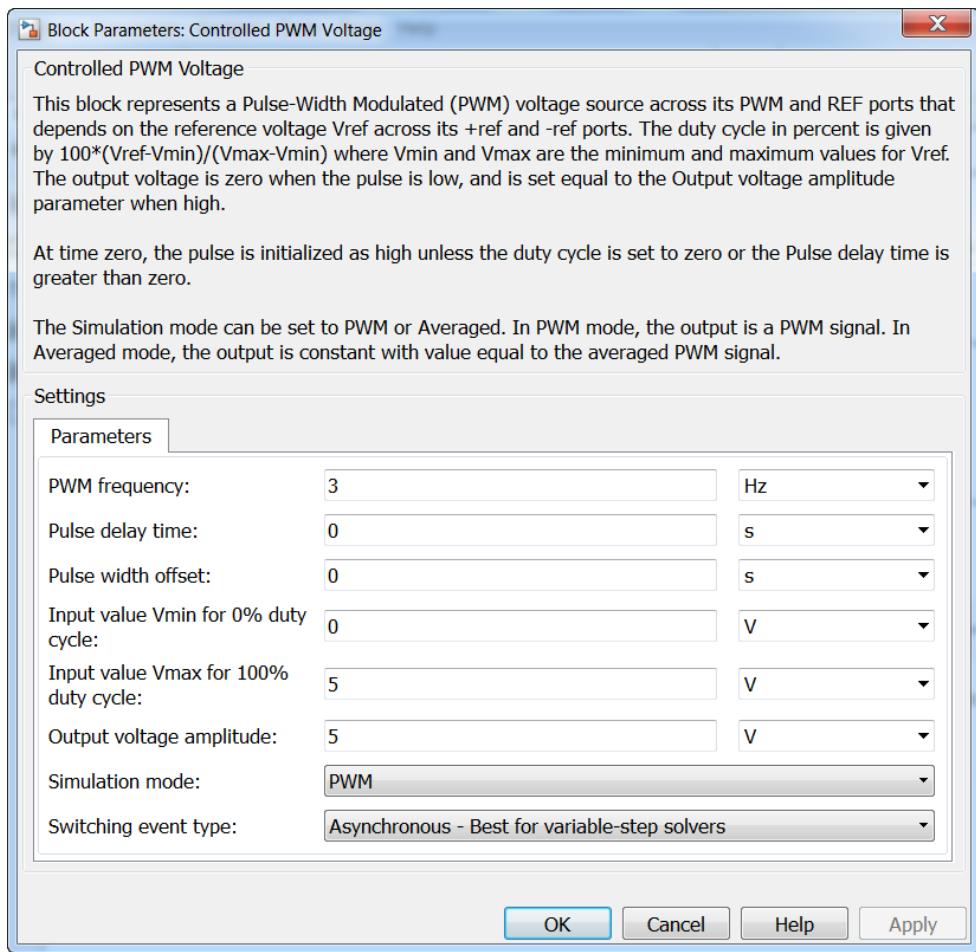


Figure 156: configuration of the Controlled PWM Voltage Source block

PWM frequency: frequency of the PWM signal

Pulse delay time: enables the start-up time of the generation of the PWM signal to be delayed

Pulse width offset: enables an offset to be imposed on the signal frequency

Input value Vmin for 0% duty cycle: voltage value on the +ref port that gives a duty cycle of 0%.

Input value Vmin for 100% duty cycle: voltage value on the +ref port that gives a duty cycle of 100%.

Output voltage amplitude: amplitude of the PWM voltage.

Simulation mode: it is possible to choose an outgoing **PWM** or **Averaged** type signal that averages the PWM signal and enables limiting the solver's calculation time.

2. PWM control for a DC motor

A direct current may be controlled using the “Controlled PWM Voltage Source” block.

Open the **PWM_motor_command_US.slx** file.

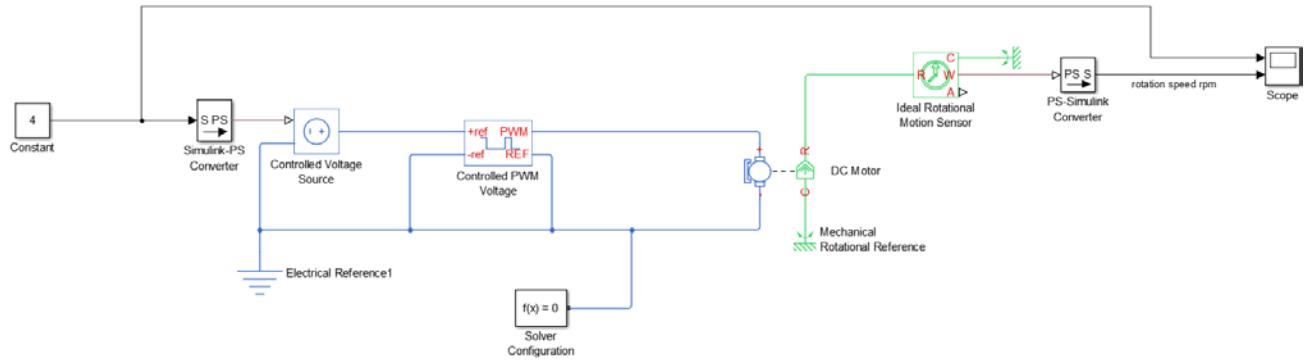


Figure 157: PWM control for a DC motor

This model represents a DC motor controlled by a PWM control voltage.

Launch the simulation and observe the rotation speed of the motor using the scope (Figure 158).



Figure 158: rotation speed of the motor

The frequency of the PWM voltage is very low (10 hz). The signal period is too close to the time constant of the motor, which explains the “choppiness” of the speed. To obtain a more satisfactory motor speed response, the PWM signal frequency must be increased by modifying the configuration of the “Controlled PWM Voltage Source” block.

Adjust the **PWM frequency** of the block to **1000 hz** and relaunch the simulation.

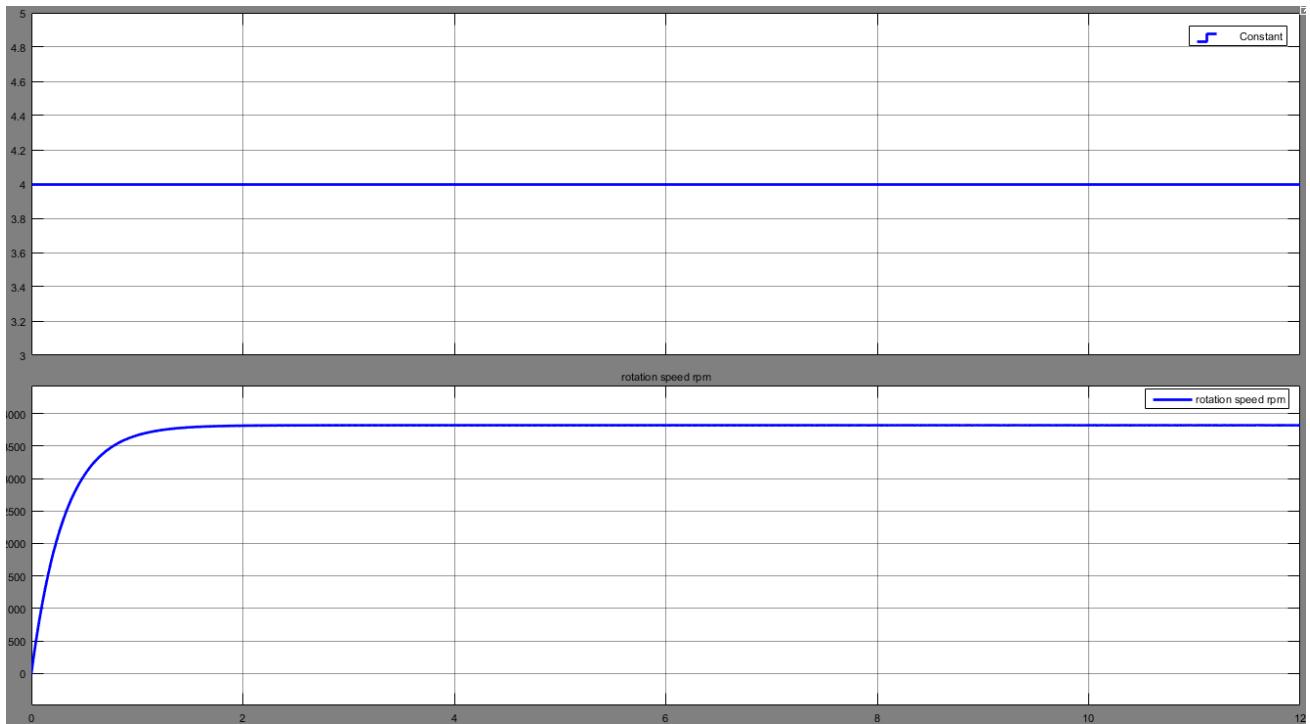
Observe the rotation speed of the motor.

Figure 159: rotation speed of the motor after changing the PWM frequency

We see in Figure 159 that the speed response is now smoother and the effect on the PWM control is less visible.

We also see that the calculation time of the solver has increased considerably. If we adjust the PWM frequency to 20 kHz, the simulation time becomes too long as the solver must perform at least 20,000 calculation steps per second. In order to simplify the work of the solver, we can set the **Simulation mode to Averaged** to control the motor with the average voltage value. For increased frequencies, the results will be identical and the calculation time greatly decreased.

Set the **Simulation mode to Averaged** and relaunch the simulation to observe the result.



The result is the same and we are able to assess the gain in solver time to resolve the model.

3. Using of the “H-Bridge” component

To be able to control the direction of the motor, an **“H-Bridge”** component must be used. The **H-Bridge** component performs this function.

Open the **Hbridge_US.slx** file.

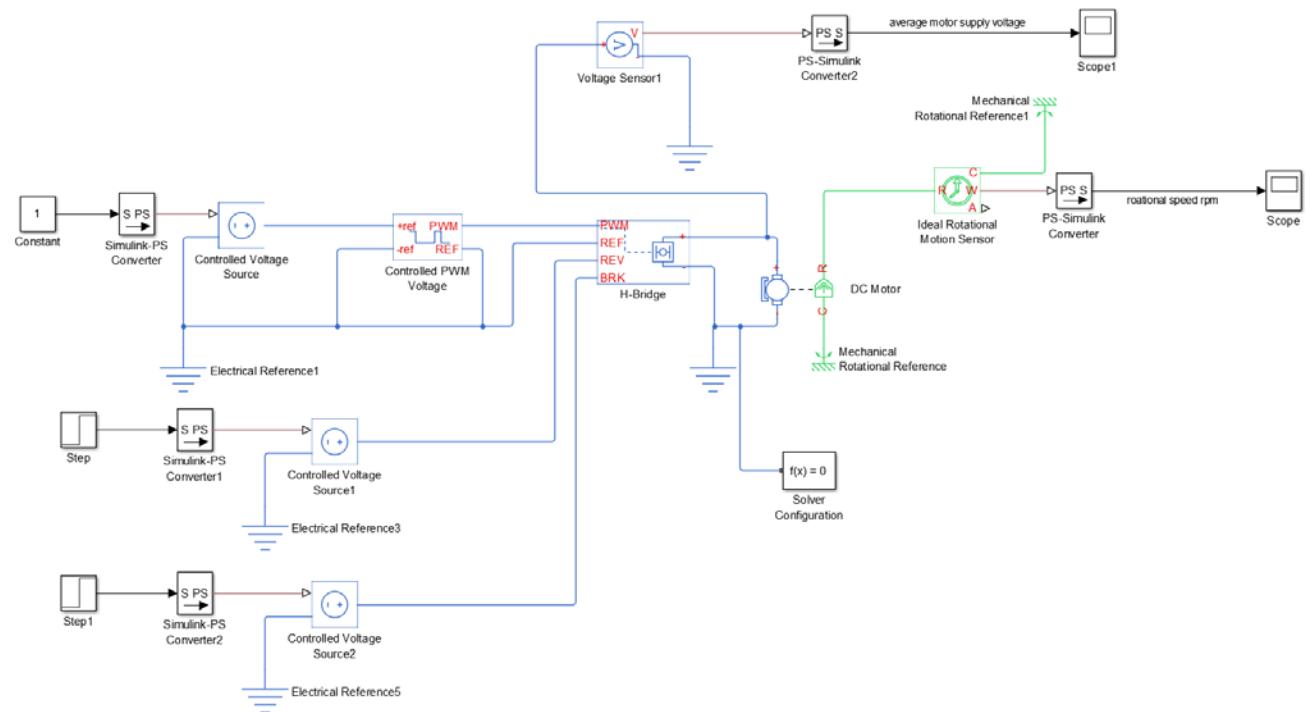


Figure 160: PWM control of a DC motor with an H-Bridge

This model enables a DC motor to be controlled using a PWM control and an H-Bridge. The rotation speed of the motor and the average supply voltage can also be visualized.

Launch the simulation and **observe** the rotation speed of the motor.

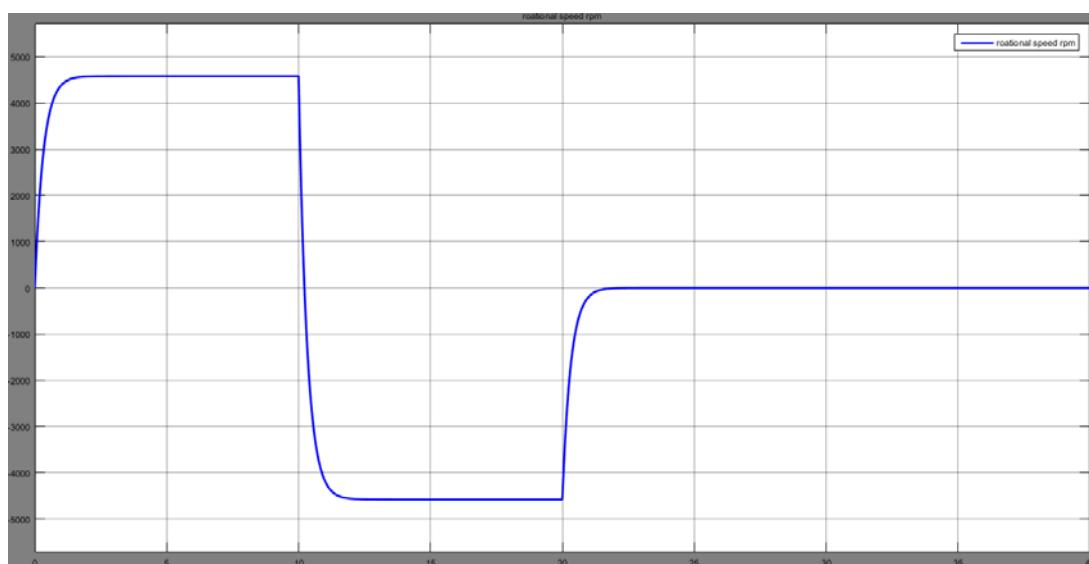


Figure 161: rotation speed of the motor controlled by the H-Bridge

The motor turns in the positive direction until $t=10\text{s}$, then in the negative direction until $t=20\text{s}$, then stops from $t=20\text{s}$.

We note that at the moment of $t=10\text{s}$, a voltage step of 5 V is imposed on the **REV** input of the H-Bridge, causing the reversal of the direction of rotation. At $t=10\text{s}$, a voltage step of 5 V is imposed on the **BRK** input of the H-Bridge, causing the motor to stop.

Observe the changes in the average voltage of the motor power supply.

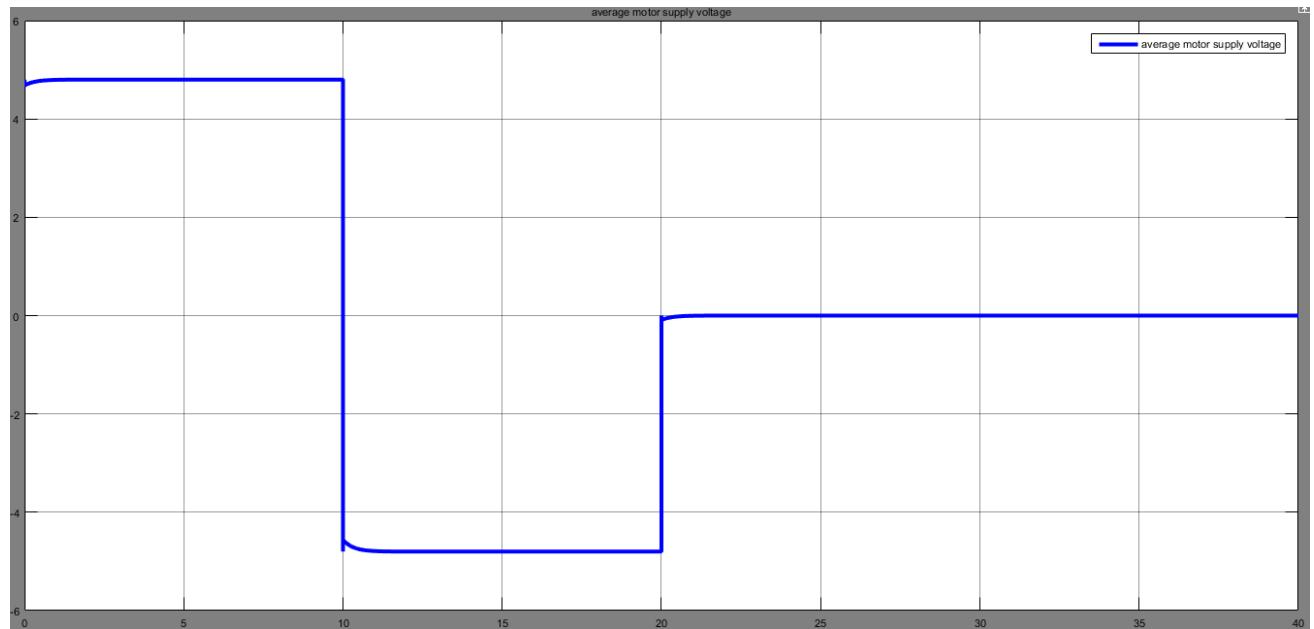
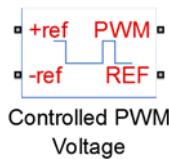


Figure 162: average voltage of the motor power supply

Configuration

Controlled PWM Voltage Source



Simscape/SimElectronics/Actuators and Drivers/Drivers

The PWM frequency is regulated to 1000hz and **Simulation mode** to **Averaged**.

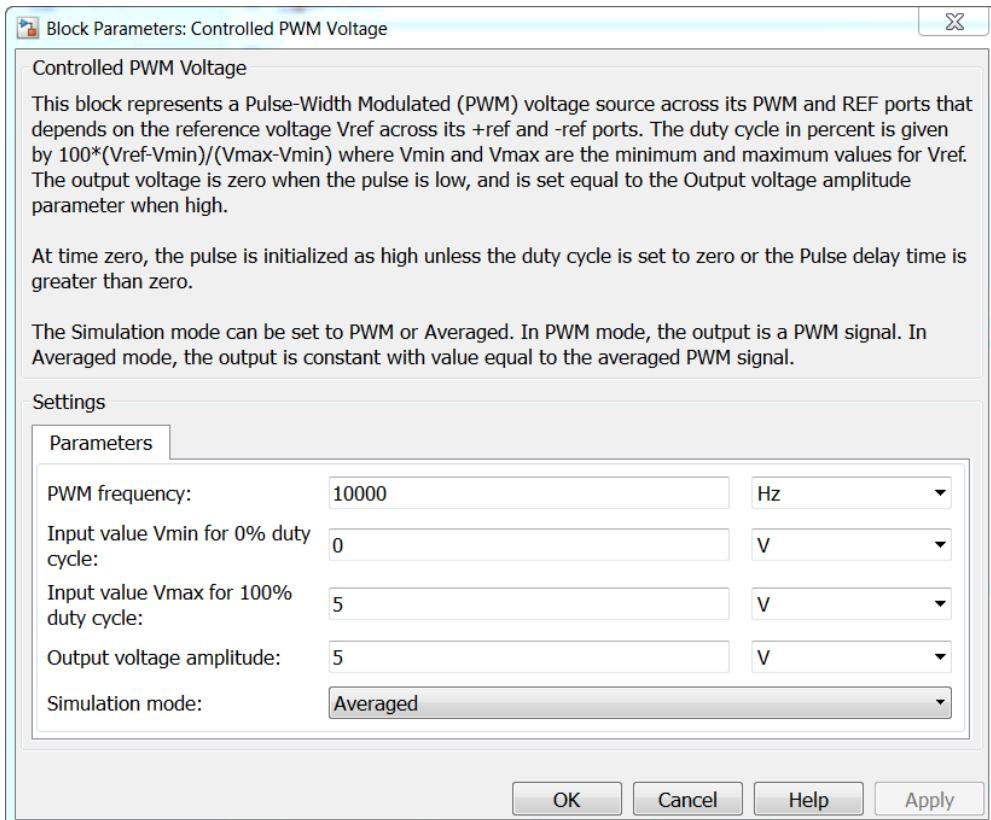
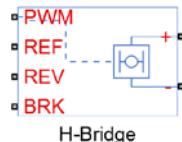


Figure 163: configuration of the Controlled PWM Voltage block

Configuration

H-Bridge



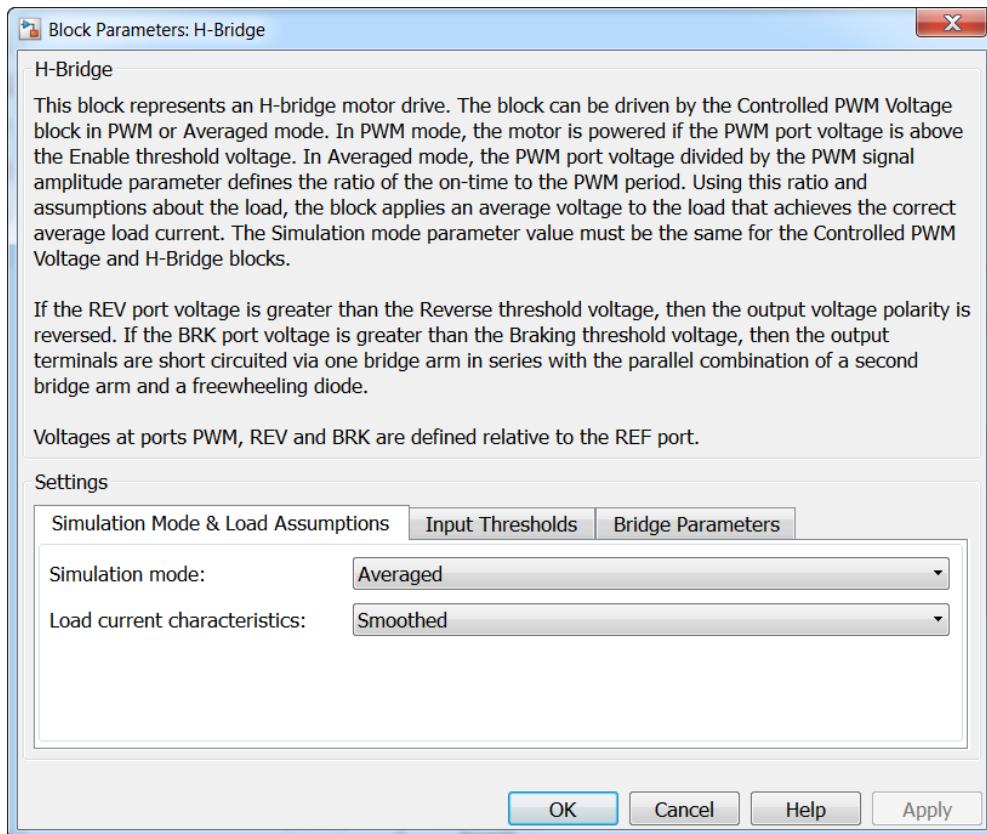
Simscape/SimElectronics/Actuators and Drivers/Drivers

This component models an H-Bridge. It possesses 6 **PCP** type ports from the electric field.

- **PWM:** PWM entry signal coming from the Controlled PWM Voltage Source block
- **REF:** reference voltage of the PWM signal (often associated with mass)
- **REV:** command to reverse the direction of rotation
- **BRK:** command to stop the motor
- + and -: these two ports are connected to the motor clamps

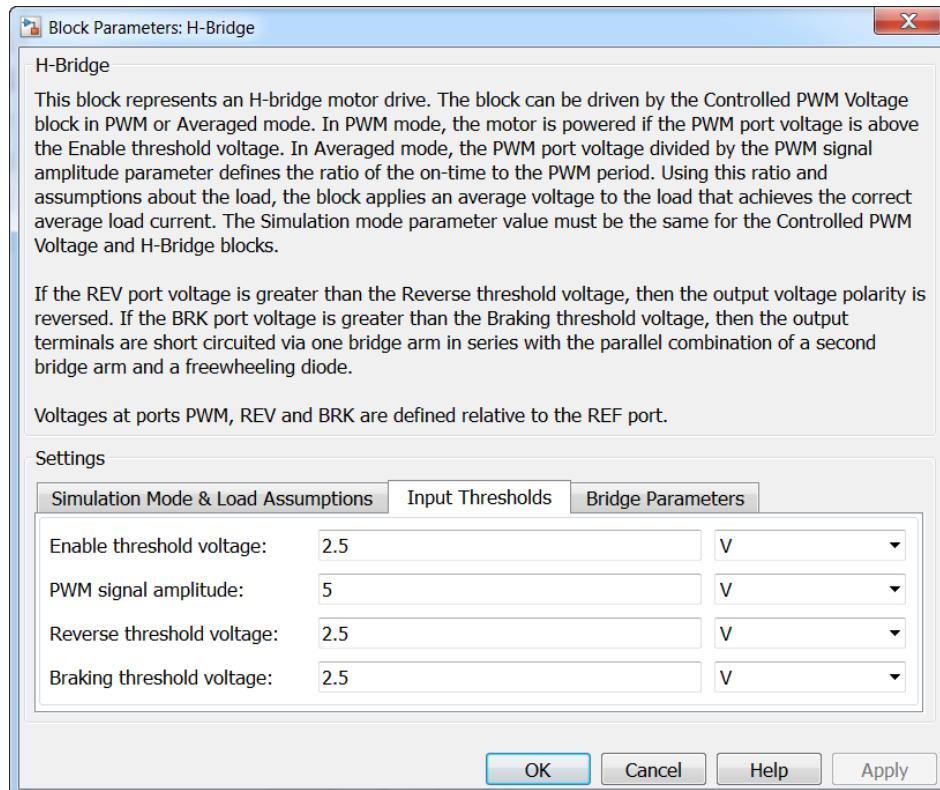
The H-Bridge receives the PWM voltage at the **PWM** input port and supplies the directly supplies the motor with this voltage. If the **REV** input voltage exceeds a set threshold in the component configuration, the direction of rotation reverses. If the **BRK** entry voltage exceeds a set threshold in the component configuration, the supply to the motor is cut off.

Simulation Mode & Load assumptions tab



Simulation Mode is set to **Averaged** to minimize the solver's calculation time.

Input Thresholds tab



This is where various thresholds that characterize the functioning of the H-Bridge are entered and stored.

Enable thresholds voltage: the level of voltage that the PWM signal amplitude must reach to trigger the supply to the motor (only valid if **Simulation mode** is set to **PWM**).

PWM signal amplitude: PWM signal amplitude that must correspond with the PWM signal amplitude generated by the "Controlled PWM voltage source" block

Reverse thresholds voltage: level of voltage indicated on the **REV** port that causes the direction of the motor voltage to reverse.

Braking thresholds voltage: level of voltage indicated on the **REV** port that causes the motor to stop.

Bridge Parameters tab

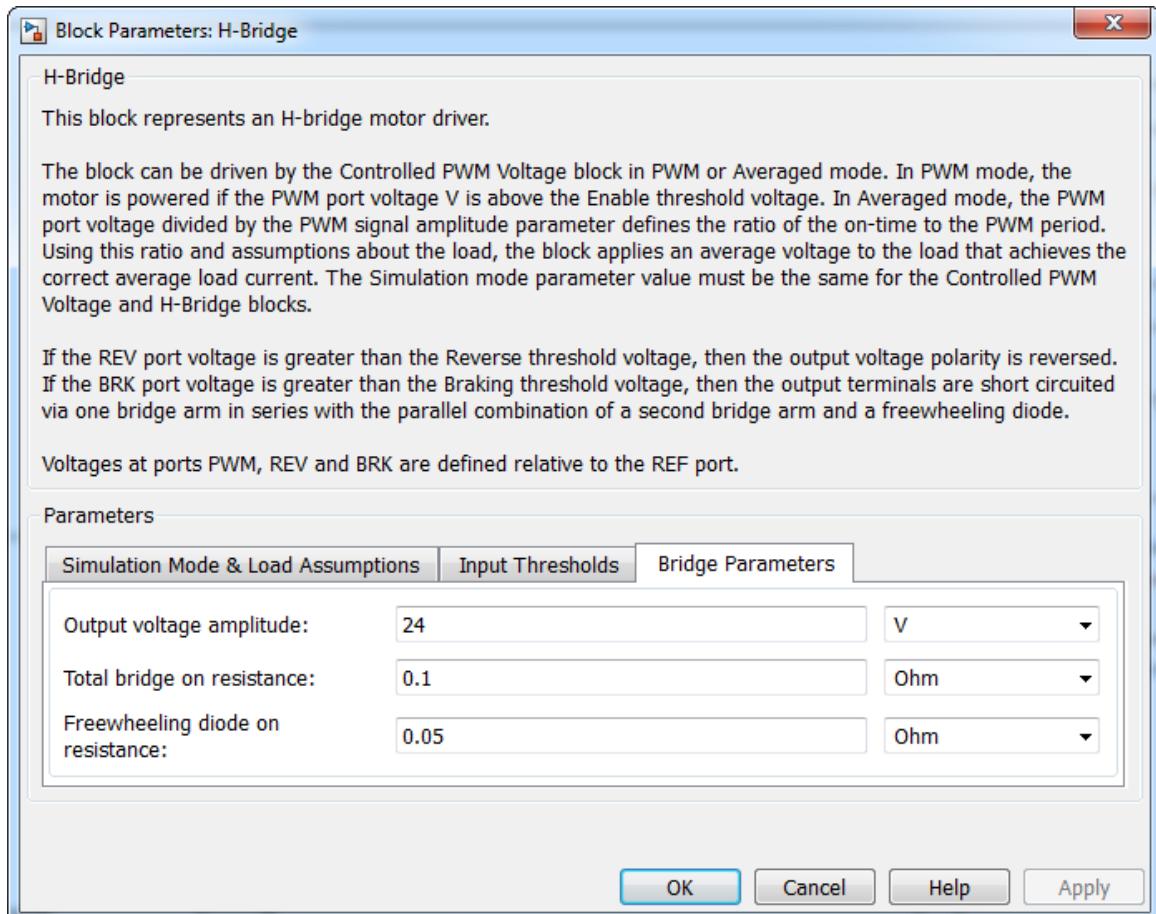


Figure 164: configuration of the H-Bridge block

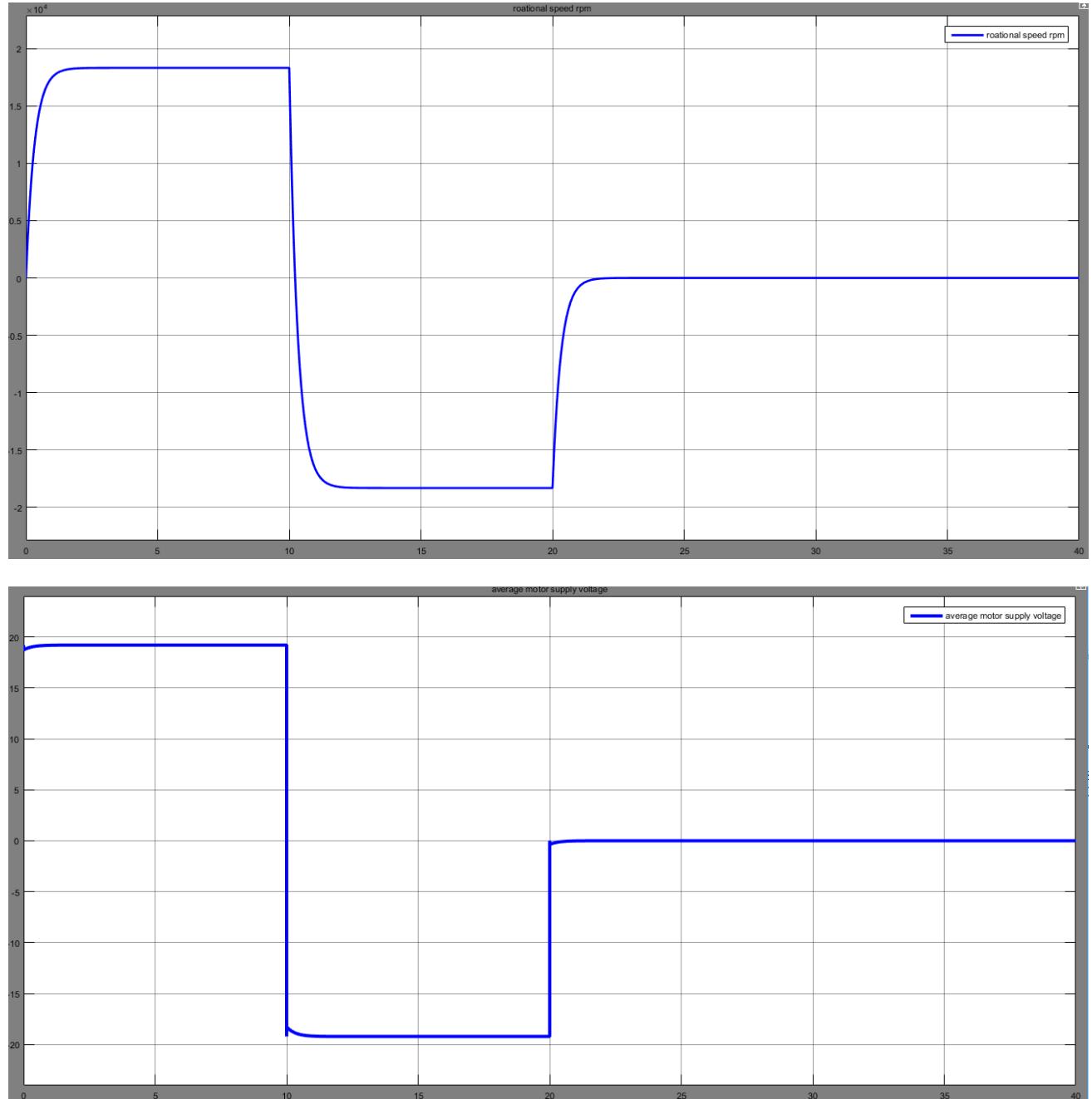
Output voltage amplitude: enables setting the maximum amplitude of the motor supply voltage to be set.

Total bridge on resistance: resistance of the two transistors that play the role of power switch for the H-Bridge.

Freewheeling diode resistance: resistance of the diodes of the bridge's freewheel that enables the transient passage of current when the power switches are closed (only for PWM Simulation Mode)

Modify the Constant block, enabling supply to be sent to the “Controlled PWM Voltage Source” and setting the value to **1** instead of **4**.

Launch the simulation and **observe** the effect on the supply voltage and the rotation speed of the motor.



D. Rendering an interactive model, using the “dashboard” library

It is possible to interact with a model during simulation. To do so, the Simulink library titled “**dashboard**” offers a series of buttons and visualization interfaces that will enable us to create interactive models (Figure 165).

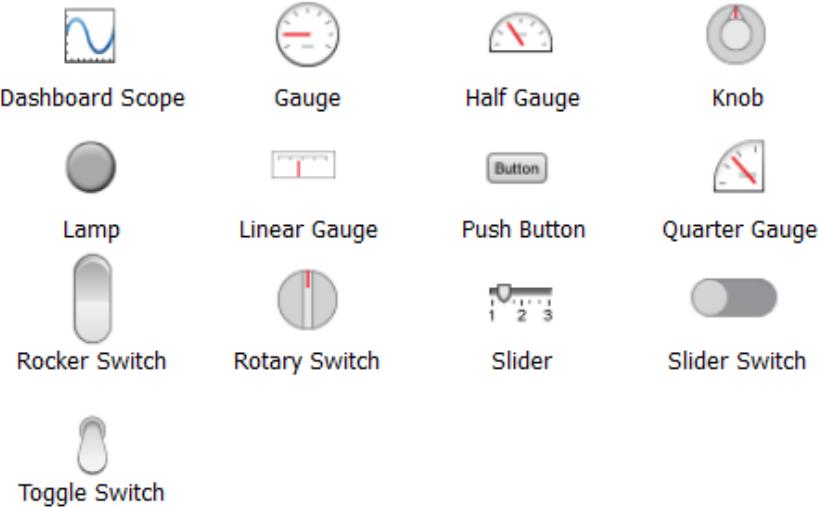


Figure 165: components of the Simulink “Dashboard” library

It is thus possible to alter the buttons to modify the model configurations and to observe the effect of this modification on the model’s behavior in real time.

To make the visualization more realistic, it is preferable that the calculation be done in real time, where 1 second in the running of the simulation will be 1 second in reality.

The “**Real-Time Pacer**” block of the **Real-Time Pacer** library enables this adjustment to be performed.

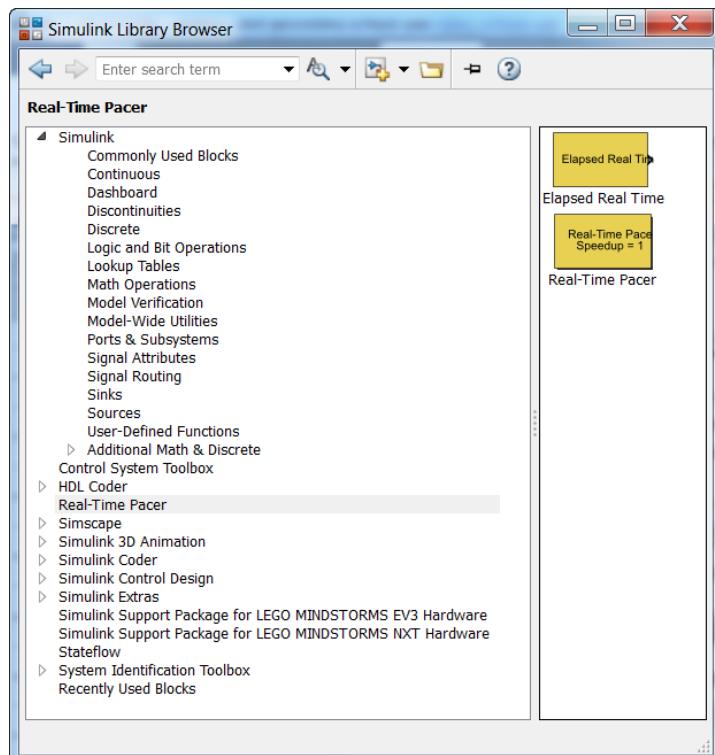


Figure 166: the Simulink Real-Time Pacer library

If the Real-Time Pacer library does not appear in the Simulink library trees, the “**Real-Time Pacer**” file must be located in the MATLAB path. **Right-click** the mouse in the Simulink library window and select **Refresh Library Browser**.

The Simulink library is updated with all of the files carrying the **.lib** extension present in the MATLAB Path.

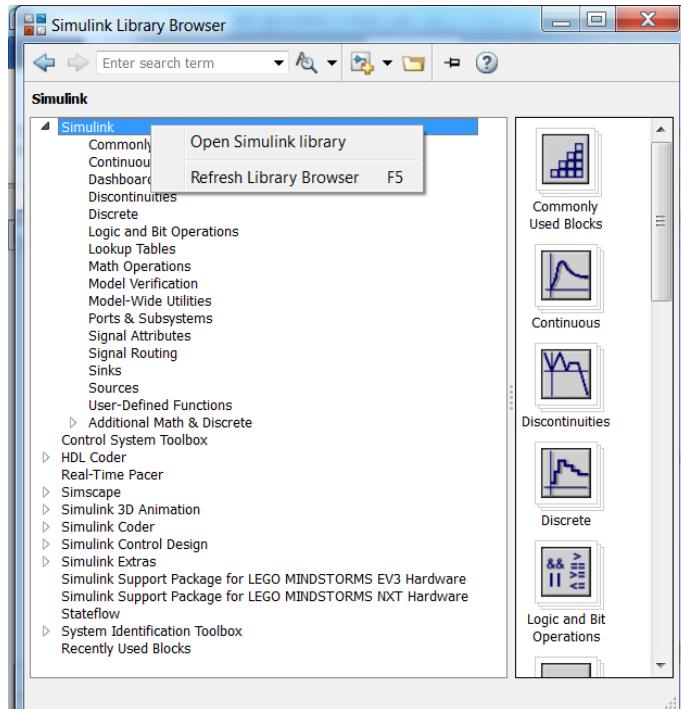


Figure 167: refreshing the Simulink library

This operation must be performed before opening the models used in this section in order for the **Real Time Pacer** block to appear.

1. Example of an interactive model

Open the “controlled_linear_axis_dashboard_US.slx” file.

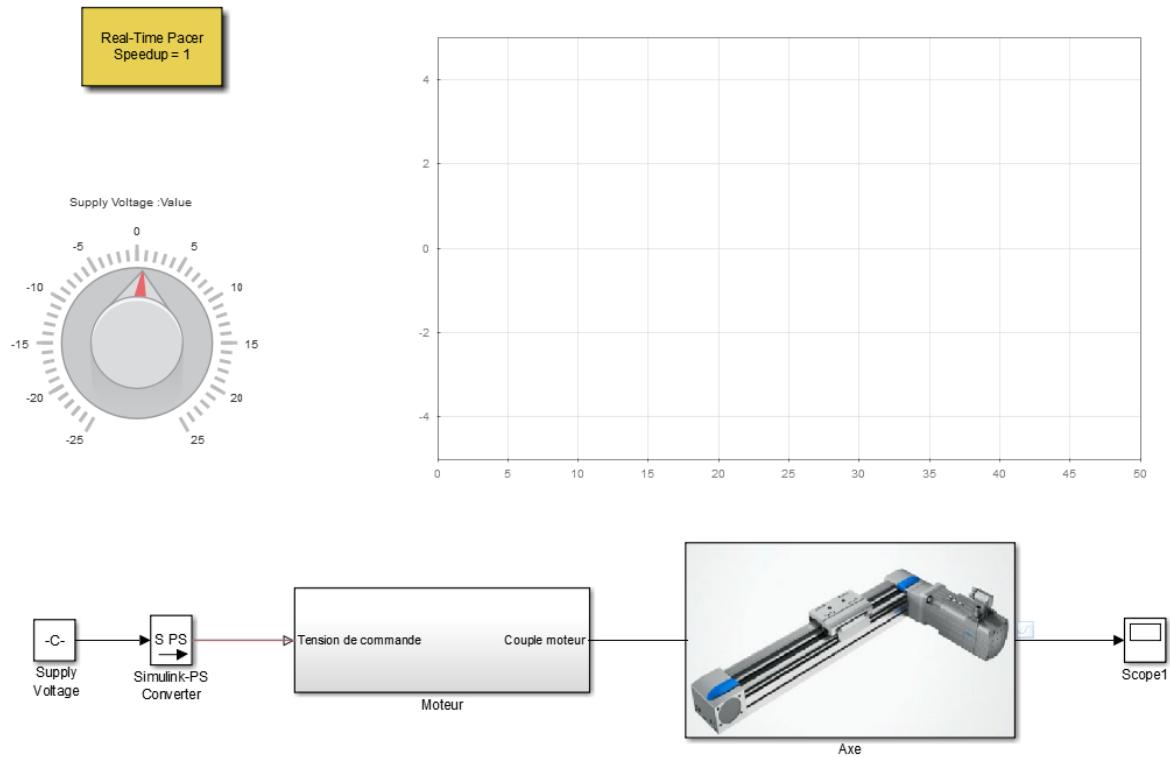


Figure 168: interactive model navigation

The **simulation time** is set to **inf**, which means that the simulation time is **infinite**.

To stop the model simulation, click the **Stop** button.

Launch the simulation and press the “**Voltage Control**” button by simply dragging and dropping, then observe the variation in the linear position of the axis directly in the interactive scope.

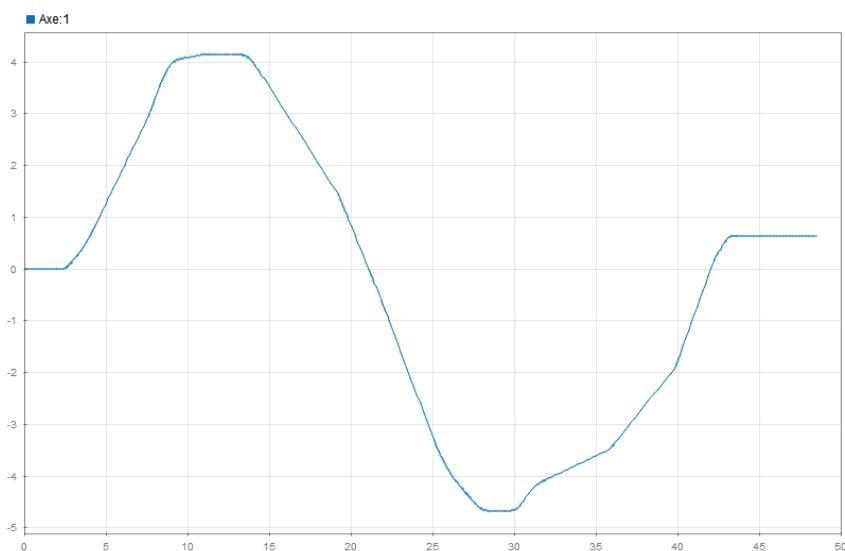


Figure 169: view of the linear axis position in real time

2. Using the library blocks

In this example we will configure the “Dashboard” library blocks of the model used previously.

Open the “controlled_linear_axis_dashboard_start_US.slx” model:

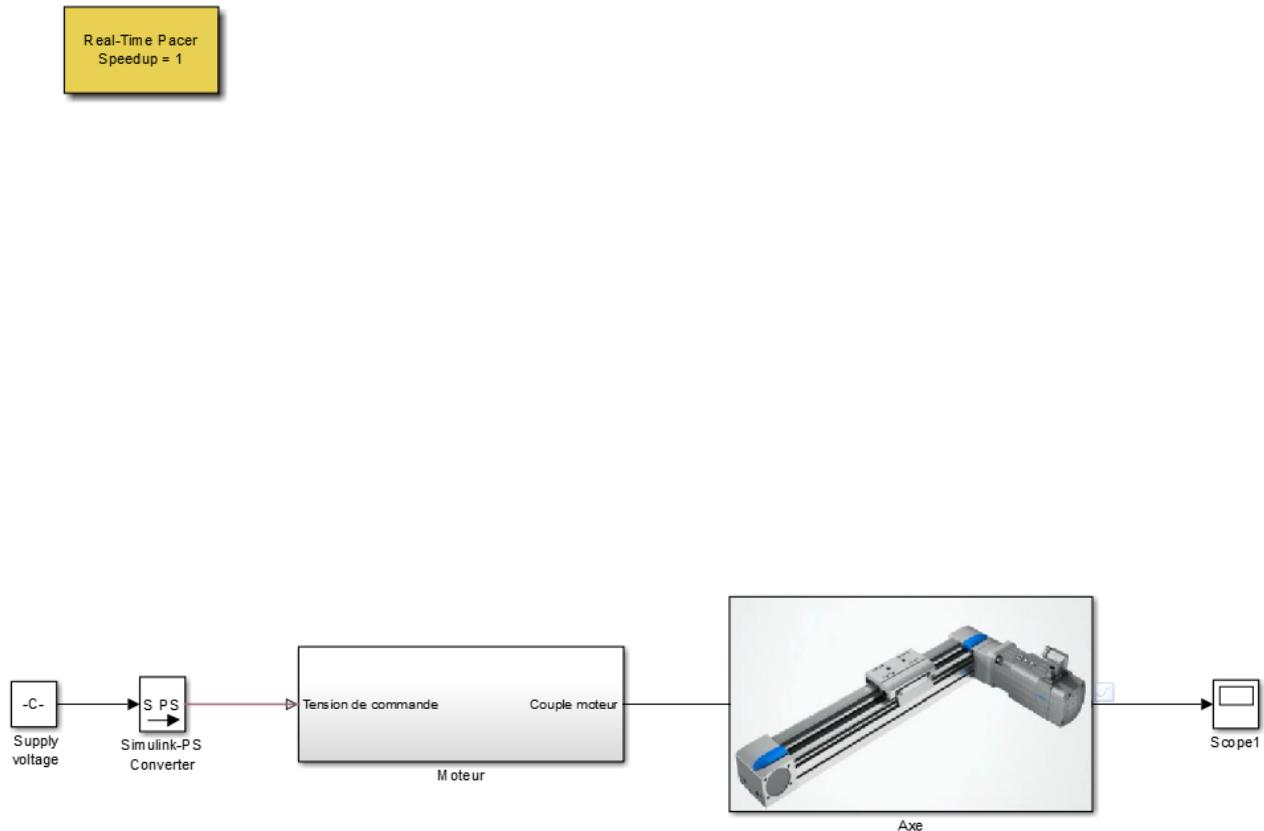


Figure 170: “controlled_linear_axis_dashboard_start_US.slx” model

Component function	View	Library
Interactive button (Knob)	 Knob	Simulink/Dashboard
Interactive scope	 Dashboard Scope	Simulink/Dashboard

Drag, position and resize the knob block and the Dashboard Scope block to obtain the configuration in Figure 171.

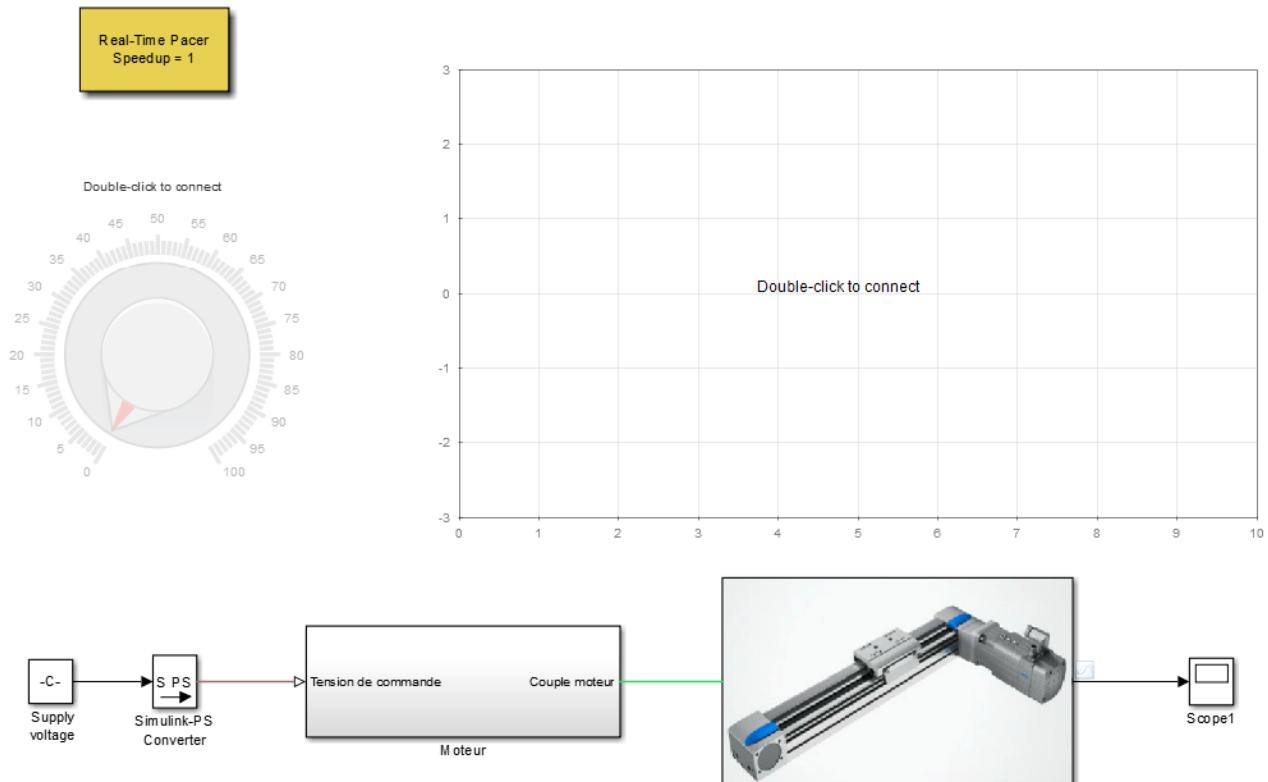


Figure 171: insertion of Dashboardlibrary blocks

At this stage, the Dashboard library blocks are not yet connected to the model and do not enable it to be interacted with.

Configuration



Double-click the Knob block.

Then **click** the block that will be controlled by the button. Here the “Supply voltage” block that sets the input voltage of the linear axis be selected, then connection must be validated by clicking the **Connect** check box (Figure 172).

The configuration of the component also includes specifying the variation range of the parameter.

Minimum: lower limit of the parameter variation

Maximum: upper limit of the parameter variation

Ticks: interval between two graduations

The configuration window should be identical to Figure 172.

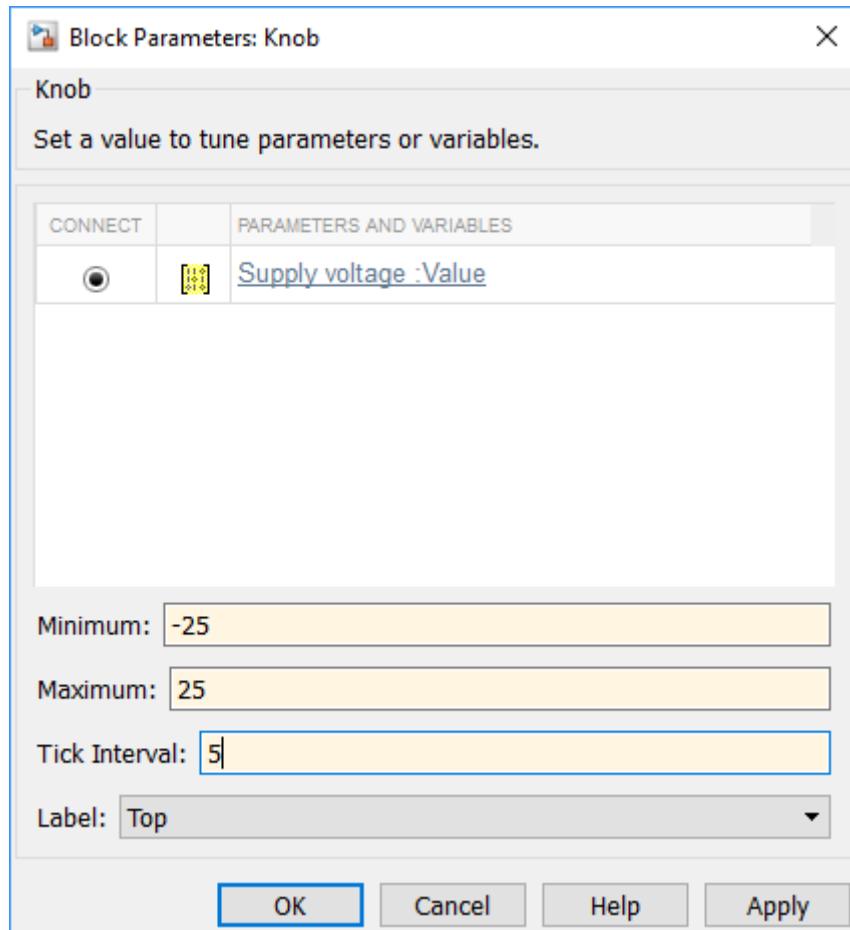


Figure 172: configuration window of the knob block

The button then changes appearance to take the form shown in Figure 173.

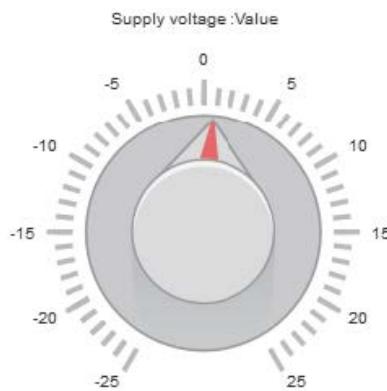
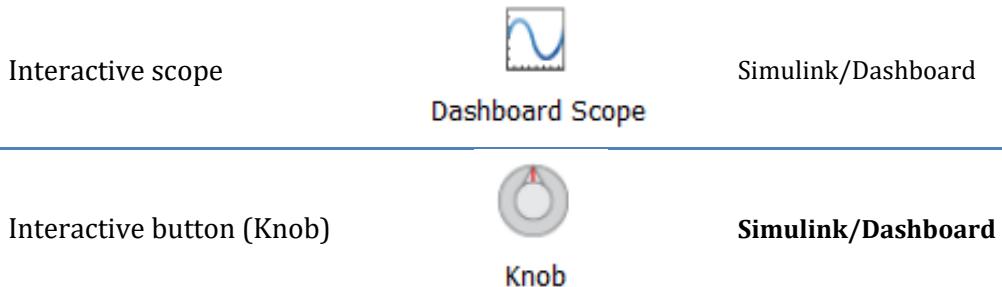


Figure 173: appearance of the knob button after configuration and connection with the model

Configuration



Double-click the Interactive Scope block.

Then **Click** the link that contains the signal that you want to display. Here the output signal of the model (which enters the scope and gives the linear position of the axis) must be selected, then validate the connection by clicking the **Connect** check box (Figure 174).

The configuration of the component also includes specifying the time span displayed for the curve, and the upper and lower limits of the Y axis.

Time Span: time range specified to observe the phenomenon

Y-Axis Limits: lower and upper limits of the Y axis

The configuration window must be identical to Figure 174.

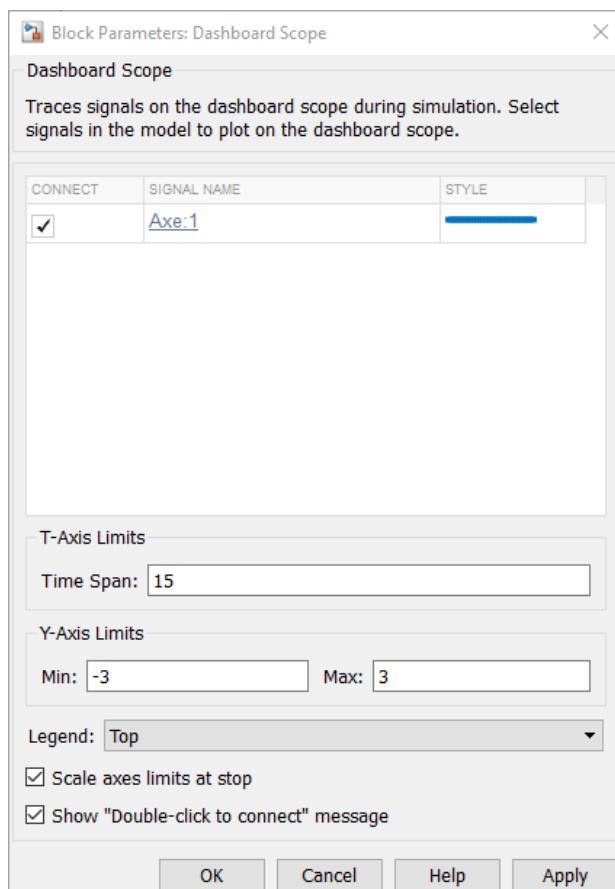


Figure 174: configuration window of the Dashboard Scope block

Launch the simulation and observe in the change of the axis position in real time.

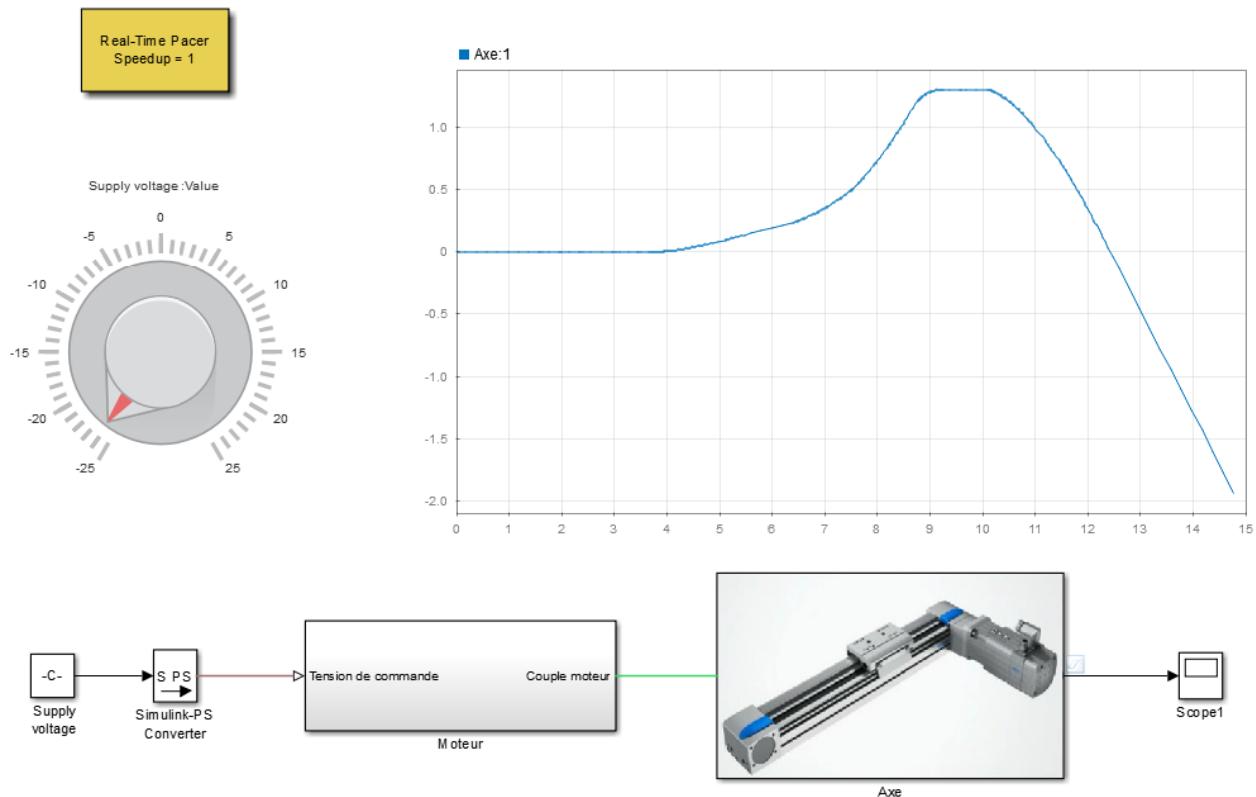


Figure 175: simulation of an interactive model with the components of the Dashboard library

V. Teaching applications

The previous sections demonstrated how to use the Simscape functionalities and libraries to build usable models. To use these models for teaching purposes, it is necessary to deepen the analysis and explain the approach used in order to use these models as training supports.

Simply presenting a model to students is not sufficient to guarantee effectiveness in the learning process. Teachers must think on how to construct a relevant course around the use of a digital model. In the example that follows, we will see how it is possible to shift from the simple design of a model to its pedagogical use with students. This approach will be illustrated by using a model of a buck converter.

A. Introduction to the buck converter

Choppers are direct converters of the DC-DC type. They have the ability to produce adjustable constant voltage from fixed constant voltage. They are used specifically for the control of direct current machines (Figure 176).

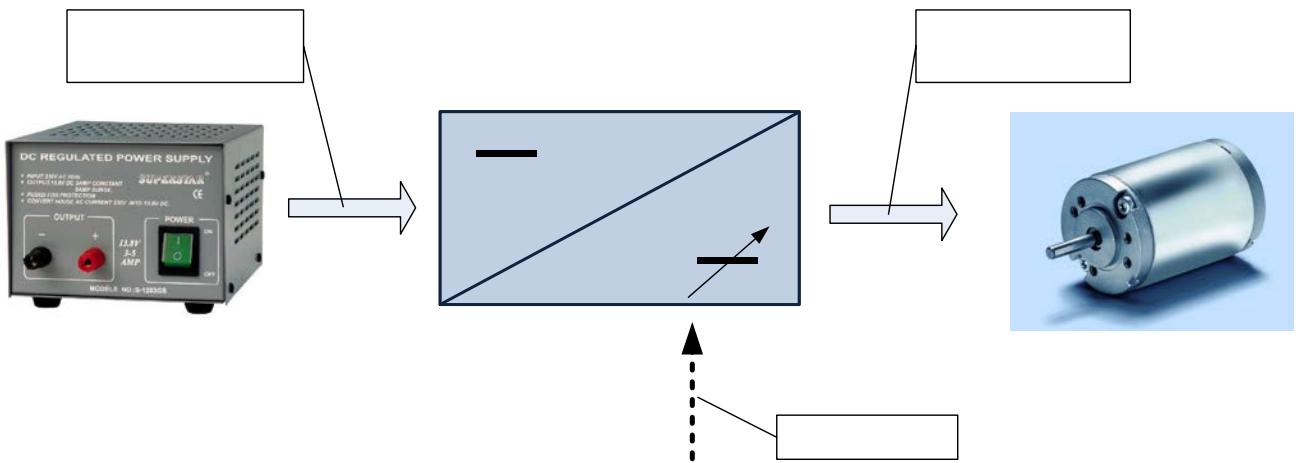


Figure 176: introduction to the buck converter

The buck converter will convert constant voltage to voltage that takes the form of square wave (Figure 177)

- T: period of chopping (or crosscutting)
- $\alpha = \frac{t_{on}}{T}$: duty cycle, if $\alpha = 1$, the outgoing voltage is direct

The signal obtained leaving the converter is called a PWM (Pulse with Modulation) signal.

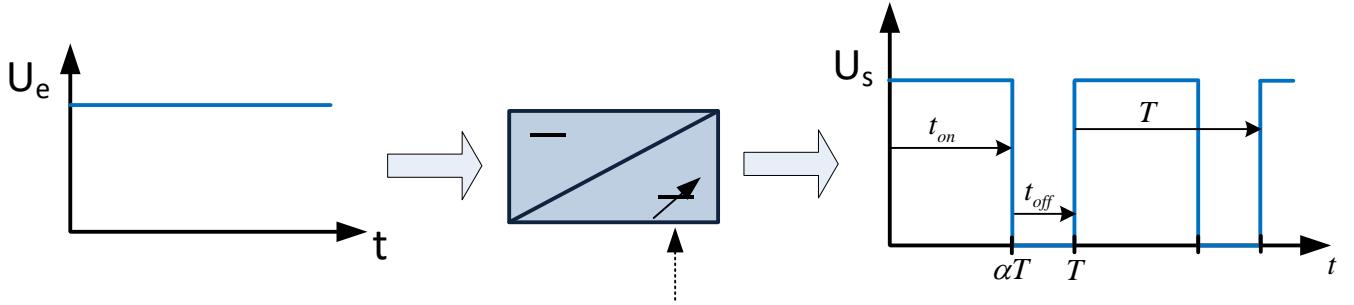


Figure 177: operating principle of the buck converter

An input source emits constant voltage \mathbf{U}_e . Periodically, a K switch is closed to obtain the voltage \mathbf{U}_s at the load terminals. In practice, the K switch is a transistor that works by commutation.

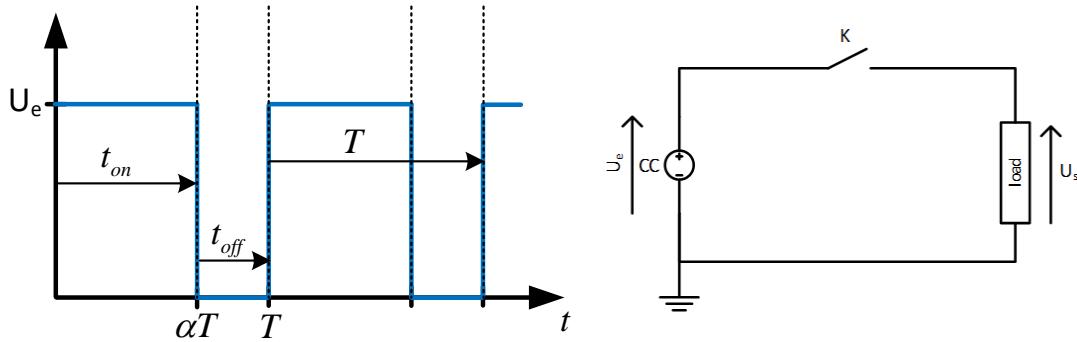


Figure 178: buck converter diagram

During a period T :

- the load is supplied over duration t_{on} , this is the active phase
- the supply is cut over duration t_{off} , this is the freewheel phase

In practice, the chopping frequency is in the order of several kHz and the load behaves as if it were supplied with an average voltage $\langle \mathbf{U}_c \rangle = \alpha \mathbf{U}_e$.

A buck converter used to control a DC motor must have 2 switches (Figure 179):

- the K1 switch is an MOS or IGBT type transistor, unidirectional in voltage and in current (controlled commutation)
- the K2 switch is a diode, unidirectional in voltage and in current (spontaneous commutation)

In the active phase, the transistor \mathbf{K}_1 is equivalent to a closed switch. The diode \mathbf{K}_2 is blocked ($V_K > V_A$) and behaves like a closed switch.

In freewheel phase, the transistor \mathbf{K}_1 is equivalent to an open switch. The diode \mathbf{K}_2 is open ($V_A = V_K$). A diode used in this manner is called a "freewheel diode".

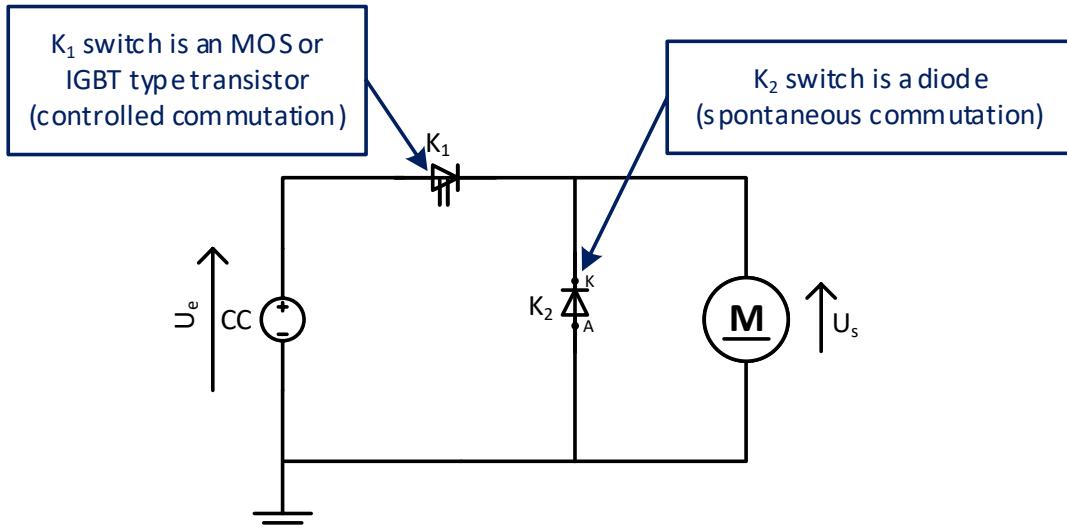


Figure 179: diagram of a current motor controlled by a buck converter

B. Teaching objectives

At the end of the session, the student should be able to understand the operating principles of the buck converter:

Objective 1: To understand the circulation of the current in the circuit in active phase and in freewheel phase (Figure 180).

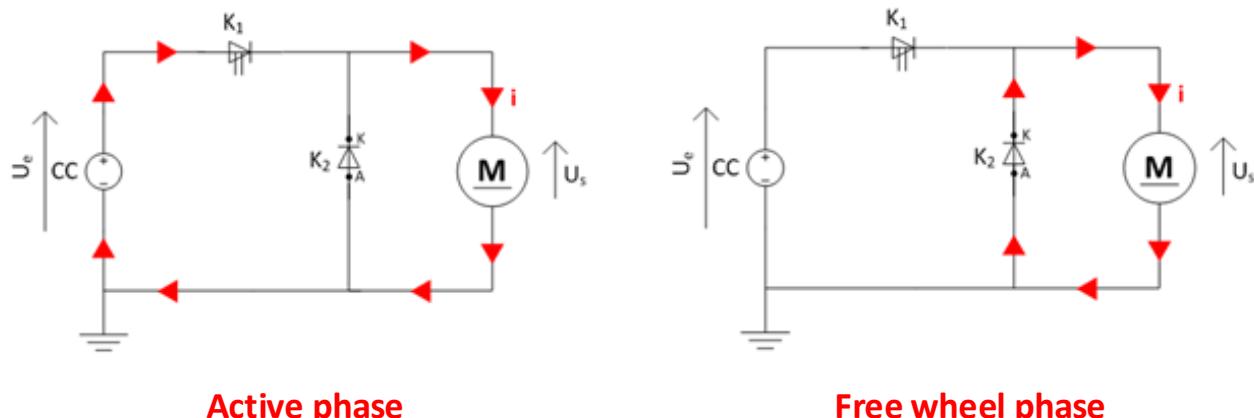


Figure 180: circulation of the current in the circuit in active phase and in freewheel phase

Objective 2: To visualize and evaluate the effect of the duty cycle on the current value (Figure 181)

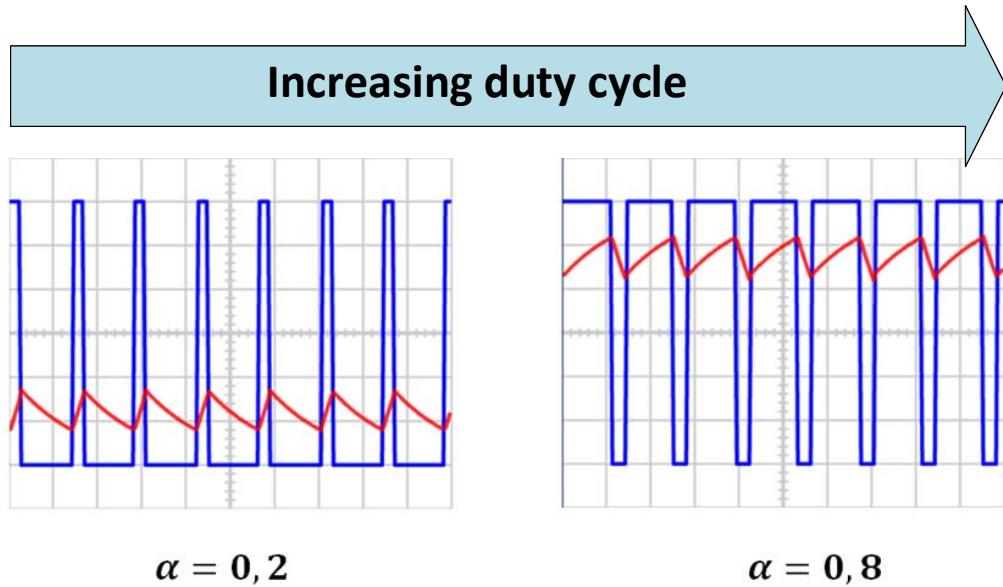


Figure 181: View of the effect of the duty cycle on the current ripple

Objective 3: To visualize and evaluate the effect of the chopping frequency on the current ripple (Figure 182)

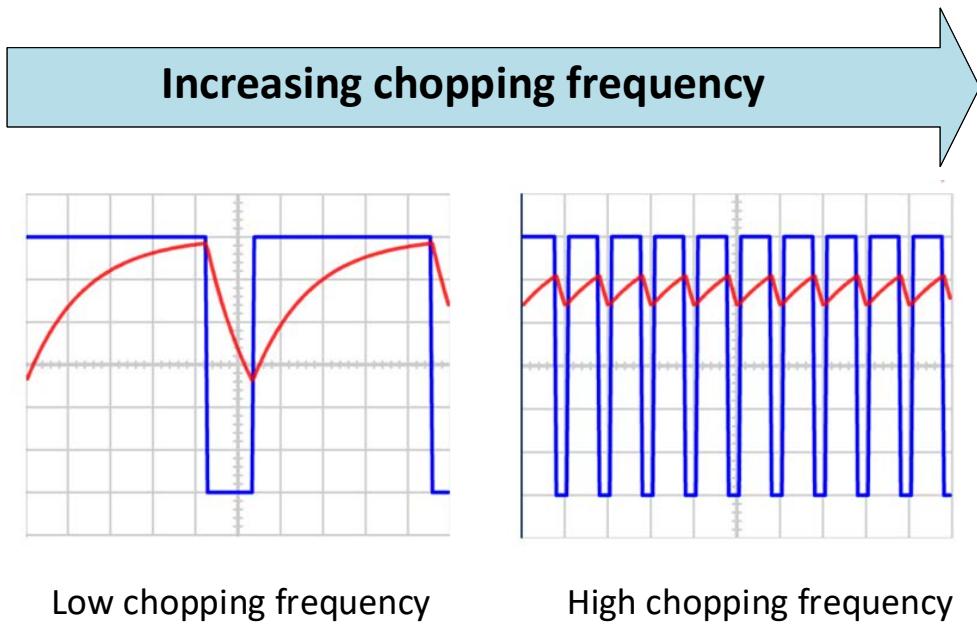


Figure 182: View of the effect of the chopping frequency on the current ripple

Objective 4: To visualize and evaluate the effect of the inductance value on the current ripple (Figure 183)

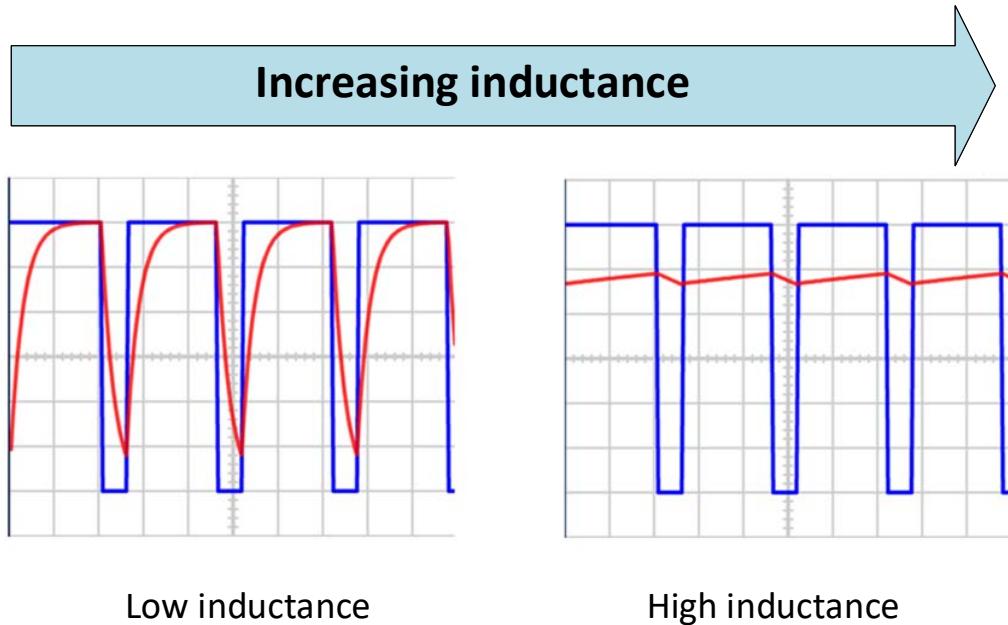
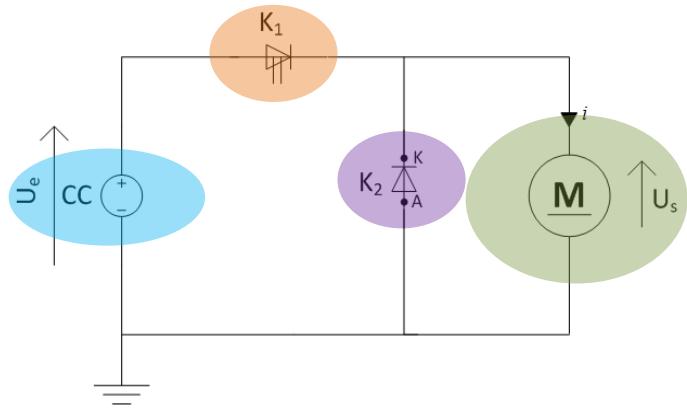


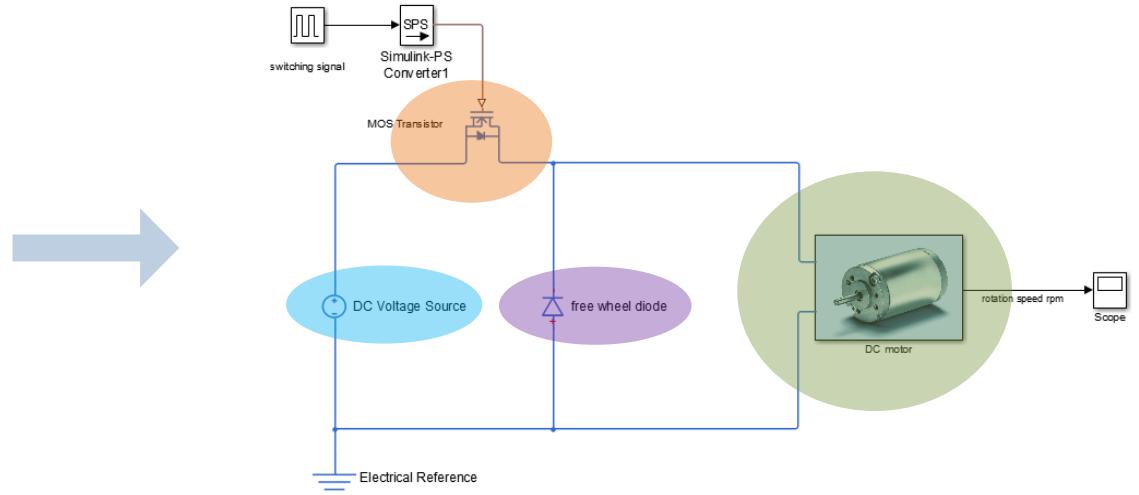
Figure 183: View of the effect of the inductance value on the current ripple

C. Constructing the model

The digital model must be built so as to create a visual analogy with the mapping in order to facilitate its use by students (Figure 184). The student easily identifies all components and can make the connection with their theoretical knowledge.



Bulk converter electrical diagram



Bulk converter Simscape model

Figure 184: analogy between the Simscape model form and the electrical diagram

Open the “bulk_converter_0_US.slx” model.

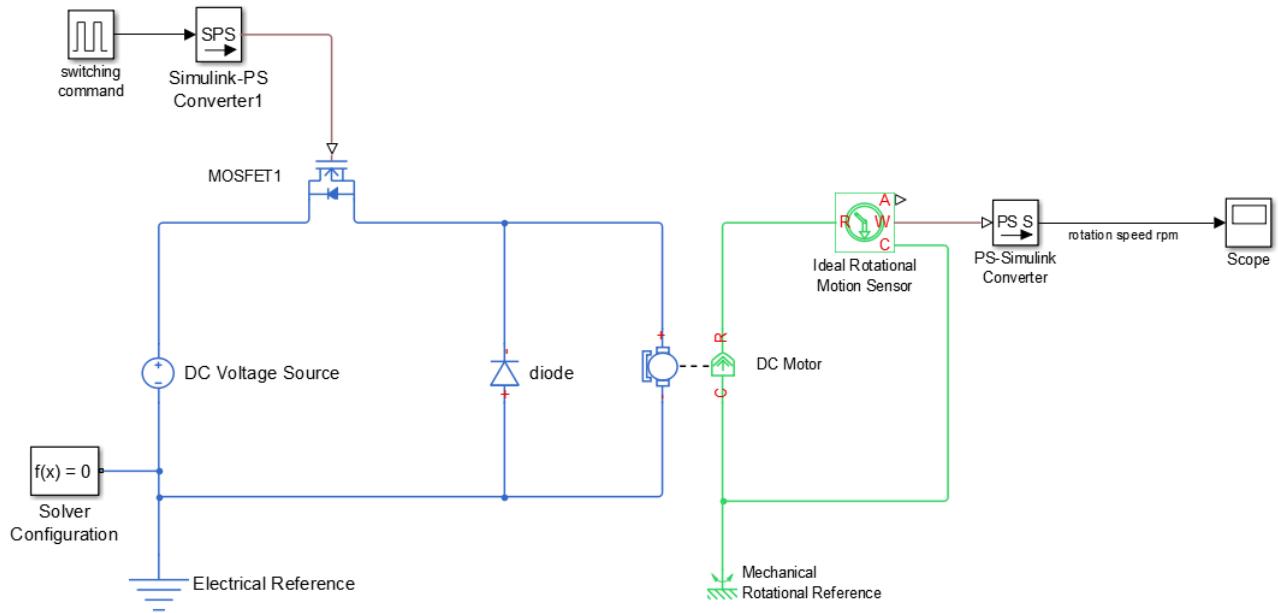


Figure 185: Simscape buck converter model

Only components that have not yet been used previously in the work are referenced in the table shown in Figure 186.

Component function	View	Library
MOSFET Transistor		Simscape/SimPowerSystems/Simscape Components/SemiConductors/Fundamental Components
Diode		Simscape/SimPowerSystems/Simscape Components/SemiConductors/Fundamental Components
Square wave generator		Simulink/Sources

Figure 186: components required for modeling the buck converter

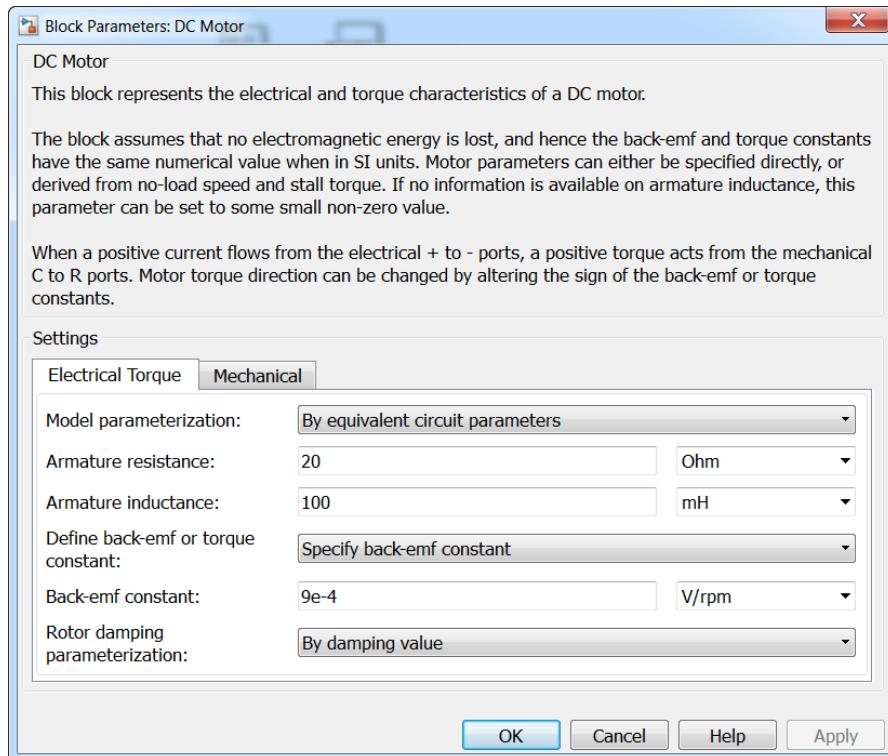
This model involves the electrical field and the mechanical rotation field.

Configuration

DC MOTOR

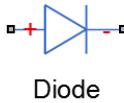


Simscape/SimElectronics/Actuators and Drivers/Rotational Actuators



Configuration

DIODE



Simscape/SimPowerSystems/Simscape Components/SemiConductors/Fundamental Components

This component is a diode. Configuration is limited to the minimal characteristics of the component.

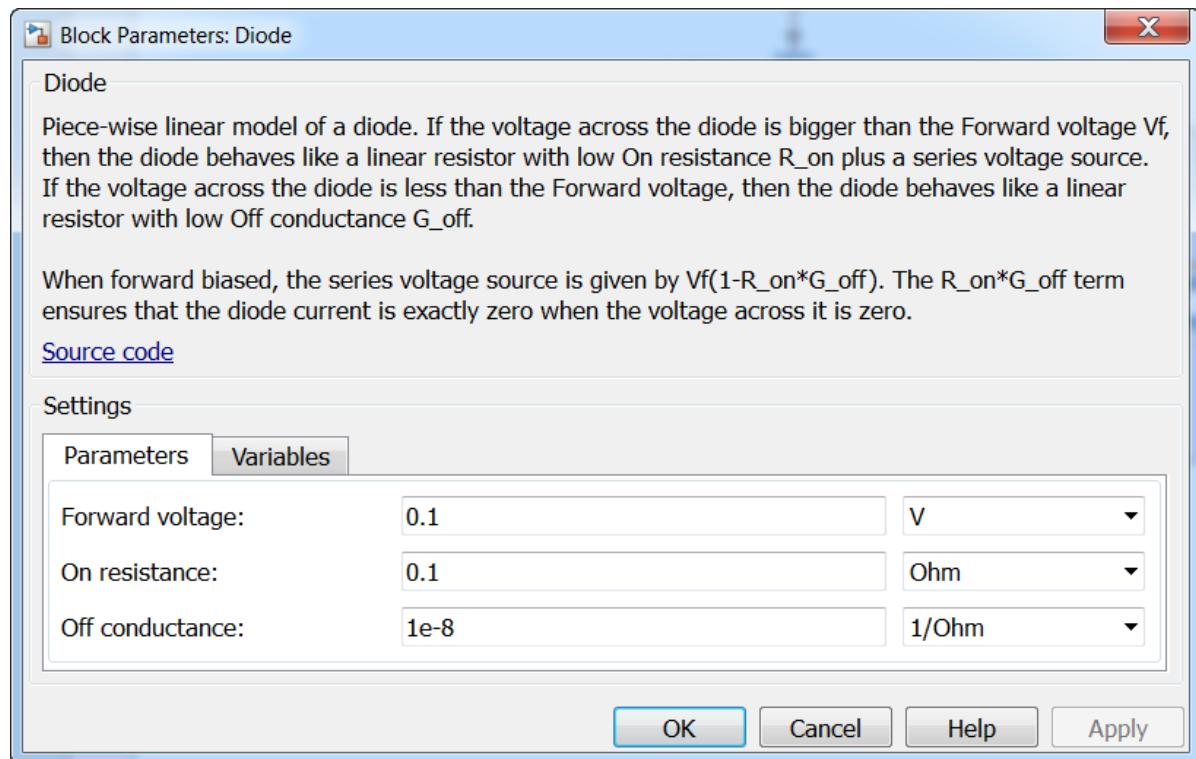


Figure 187: configuring the Diode block

Forward Voltage: minimum voltage to be applied to the diode's terminals to open it (threshold voltage)

On Resistance: resistance of the diode to the open position

Off Conductance: conductance of the diode in the closed position

Configuration

MOFSET



Simscape/SimPowerSystems/Simscape Components/SemiConductors/Fundamental Components

MOSFET

This component represents a MOFSET type transistor that works by commutation. Configuration is limited to the minimal characteristics of the component.

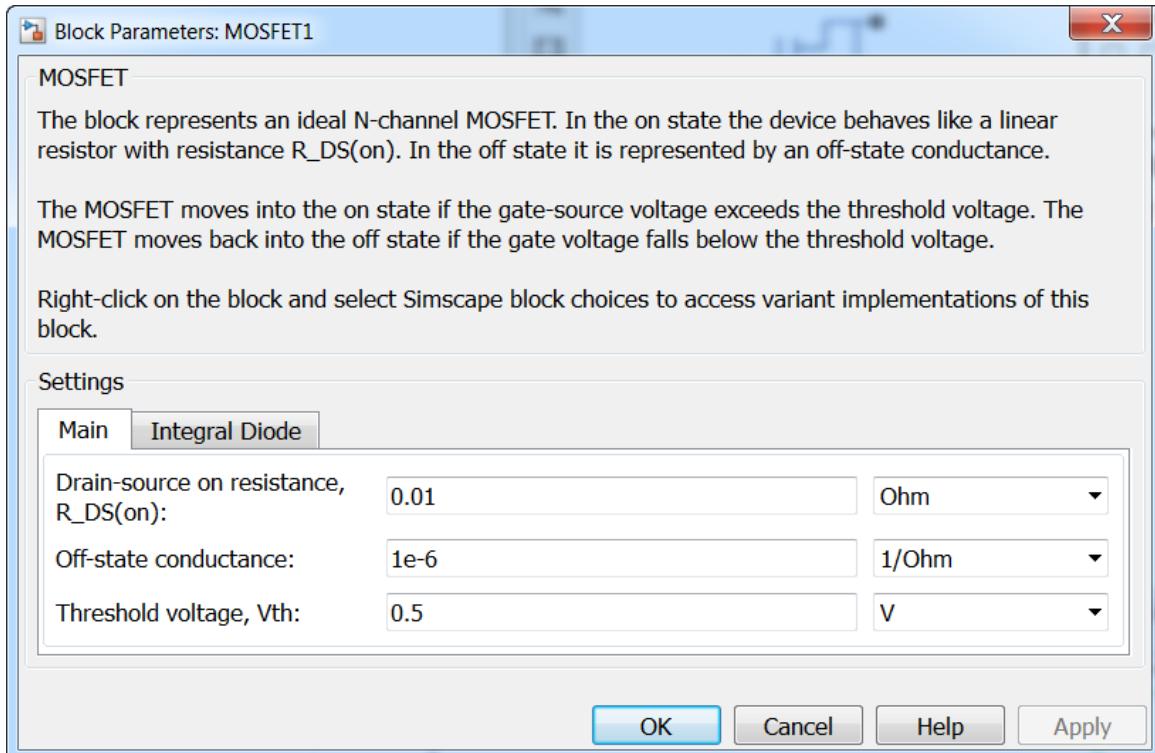


Figure 188: configuring the MOFSET block

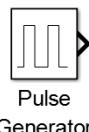
Drain-source on Resistance: resistance to the open position

Off-State Conductance: conductance to the closed position

Threshold voltage: commutation threshold voltage

Configuration

PULSE GENERATOR



Simulink/Sources

Pulse
Generator

This component enables a square wave source to be modeled by specifying the signal characteristics.

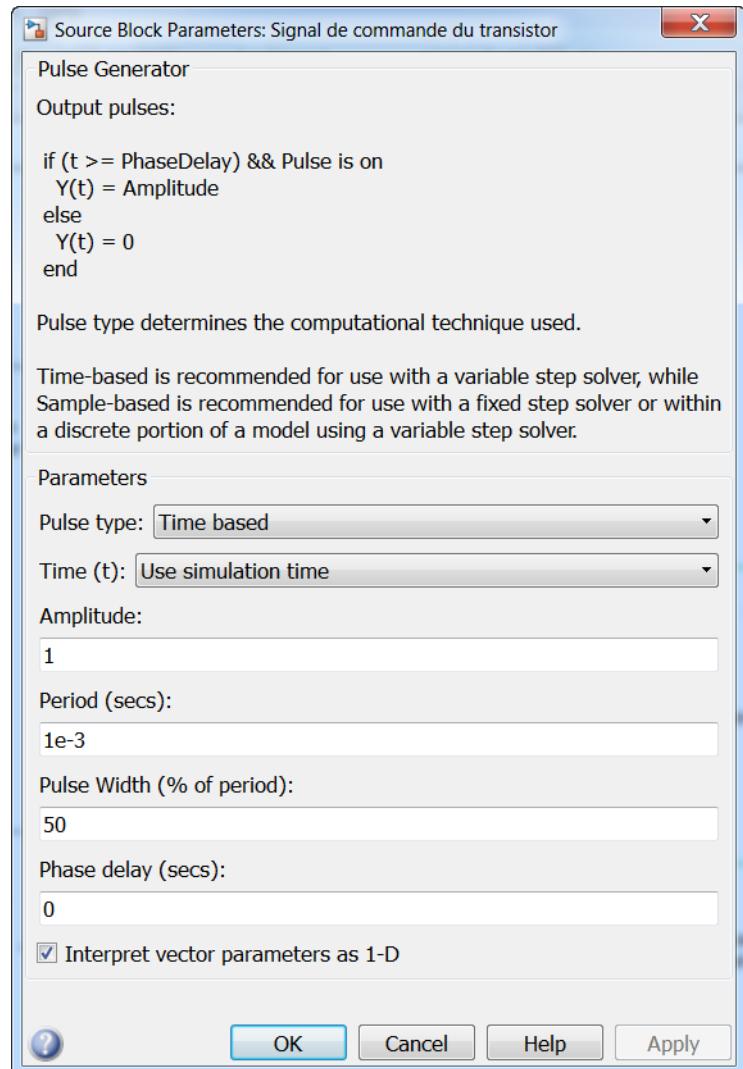


Figure 189: configuring the Pulse Generator block

Amplitude: amplitude of the square wave

Period: period of the square wave

Pulse Width: duty cycle able to vary from 0% to 100%

Phase Delay: delay

Launch the simulation and observe the change in the rotation speed of the motor using the scope.

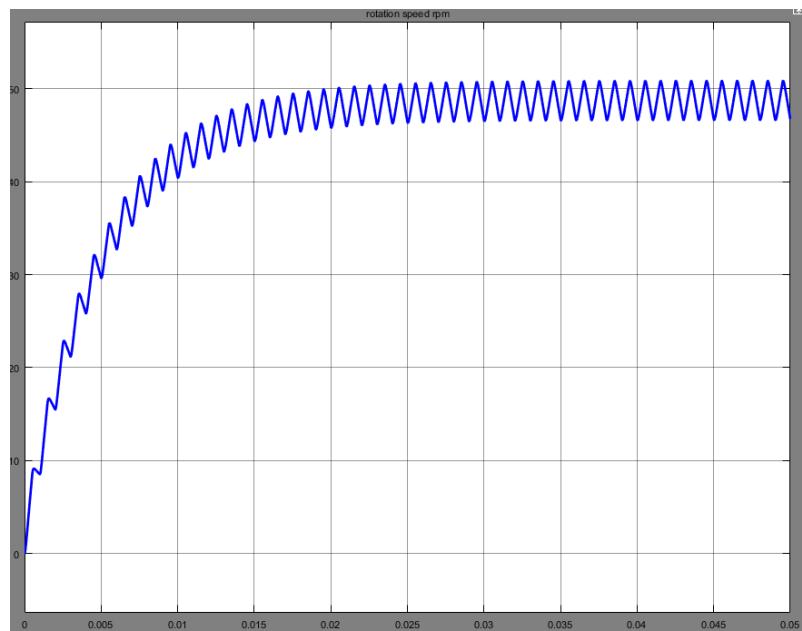


Figure 190: evolution of the rotation speed of the motor controlled by a buck converter

The chopping frequency was set here to 1 Khz and the inertia of the load was very low; the commutation frequency causes irregularities in the motor rotation speed.

Several steps will now enable us to theorize the model, that is, to render it accessible to students and make it compatible with the learning objectives.

D. The instructionalization of the model

1. Creating a sub-system and adding an image

Open the “**bulk_converter_1_US.slx**” model.

The model is the same as previously. A sub-system representing the motor was created, rendering this component more viewable. This operation must be performed whenever possible in order to improve the readability of the model.

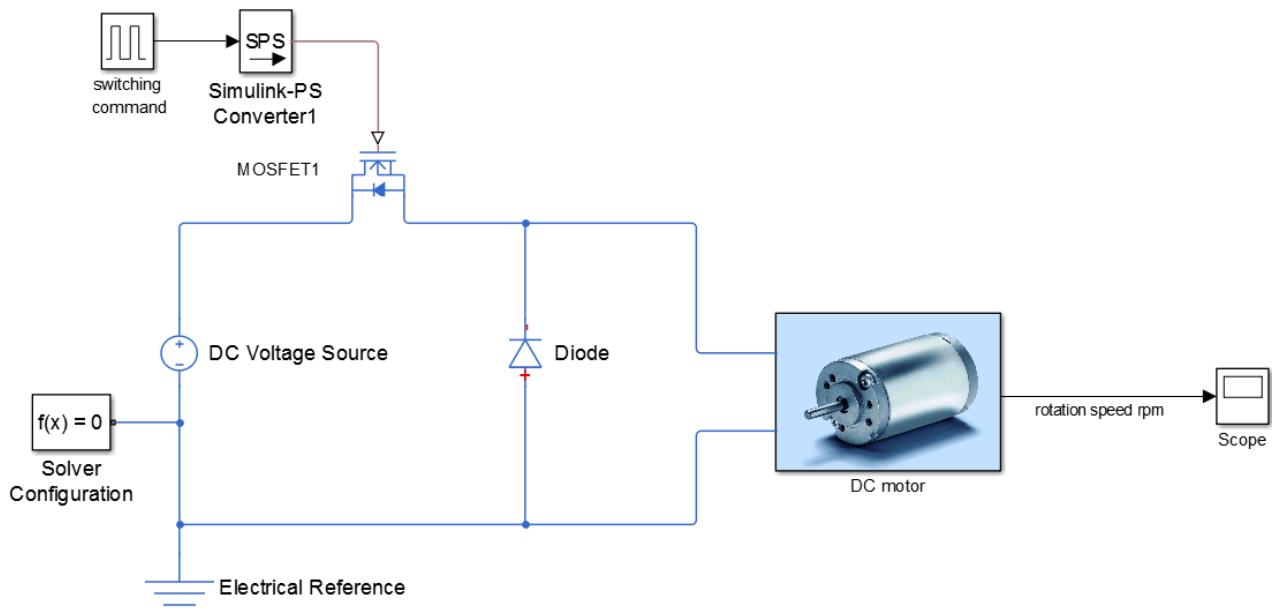


Figure 191: creating a sub-system to facilitate the model as a learning tool

2. Instrumentation of the model

One of the objectives is to help students understand how currents circulate in the converter, so we will install current sensors in each of the circuit branches (Figure 192).

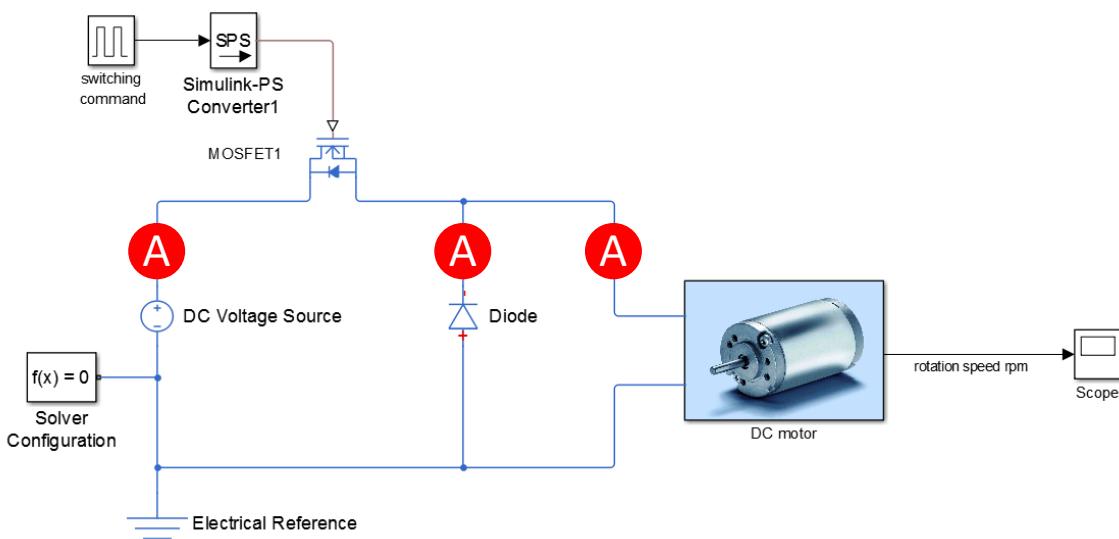


Figure 192: installing current sensors onto the circuit

We can install the first sensor to obtain the model shown in Figure 193.

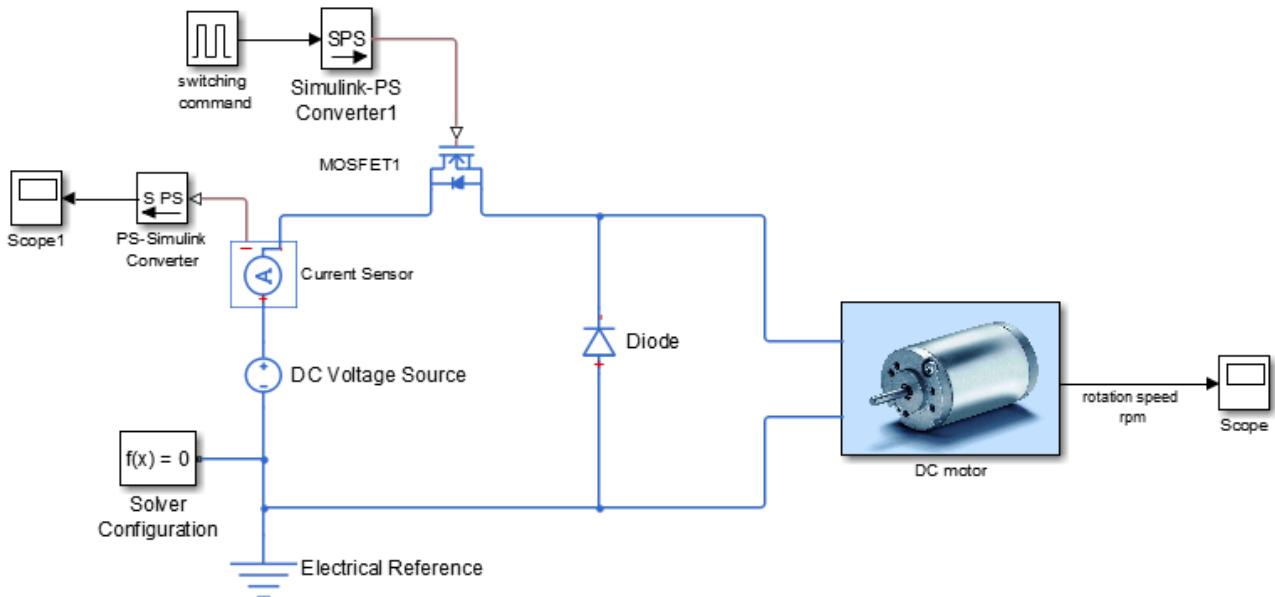


Figure 193: adding a sensor to the model

This type of instrumentation presents the major inconvenience of potentially overloading this model by adding a large number of components. This overload of components will degrade the readability of the model.

In order to optimize this phase and improve or maintain readability, it is advised that a sub-system carrying out the desired function of the signal router is built with tags by grouping them in the same scope (the functionalities of signal routing are presented on page 138.)

Open the “hacheur_serie_2.slx” file and observe the work that was performed to instrument the model.

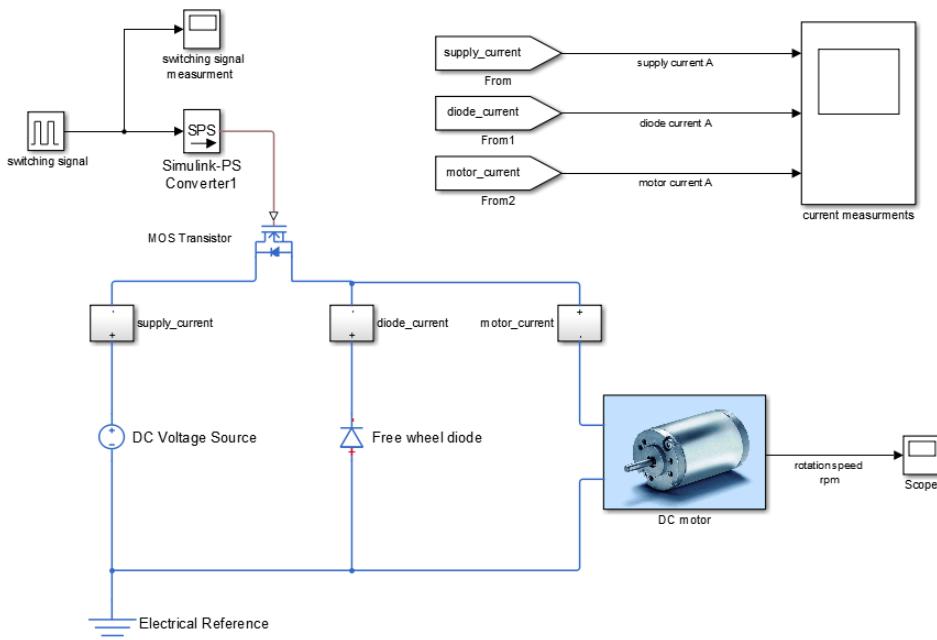


Figure 194: an instrumented model facilitated for teaching

Double-click the current sensor “Supply_current” and look at the content of the sub-system (Figure 195).

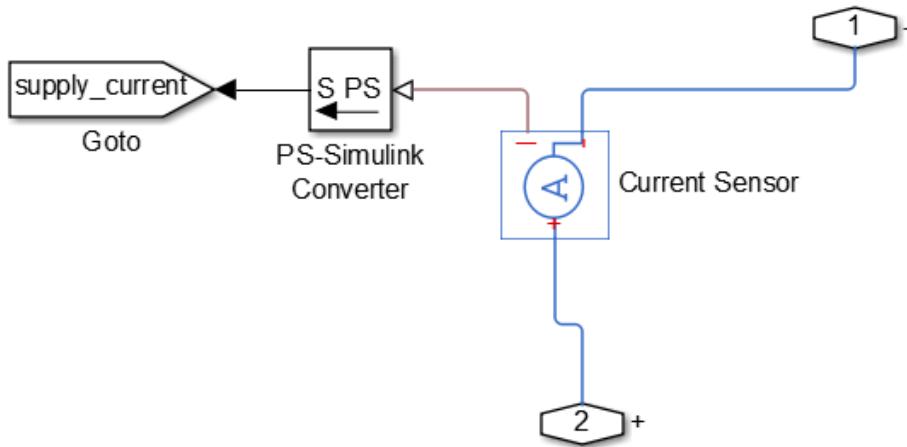


Figure 195: supply_current sub-system

This sensor enables current intensity in a branch of the circuit to be increased and for the signal to be routed using a tag that will be seen by the scope.

The advantage in this is that this structure can be copied and used in the two other branches of the circuit to minimize the addition of the new components to the model.

At this stage, the model becomes usable by the student.

Launch the simulation and view the speed of the currents in the various branches of the circuit.

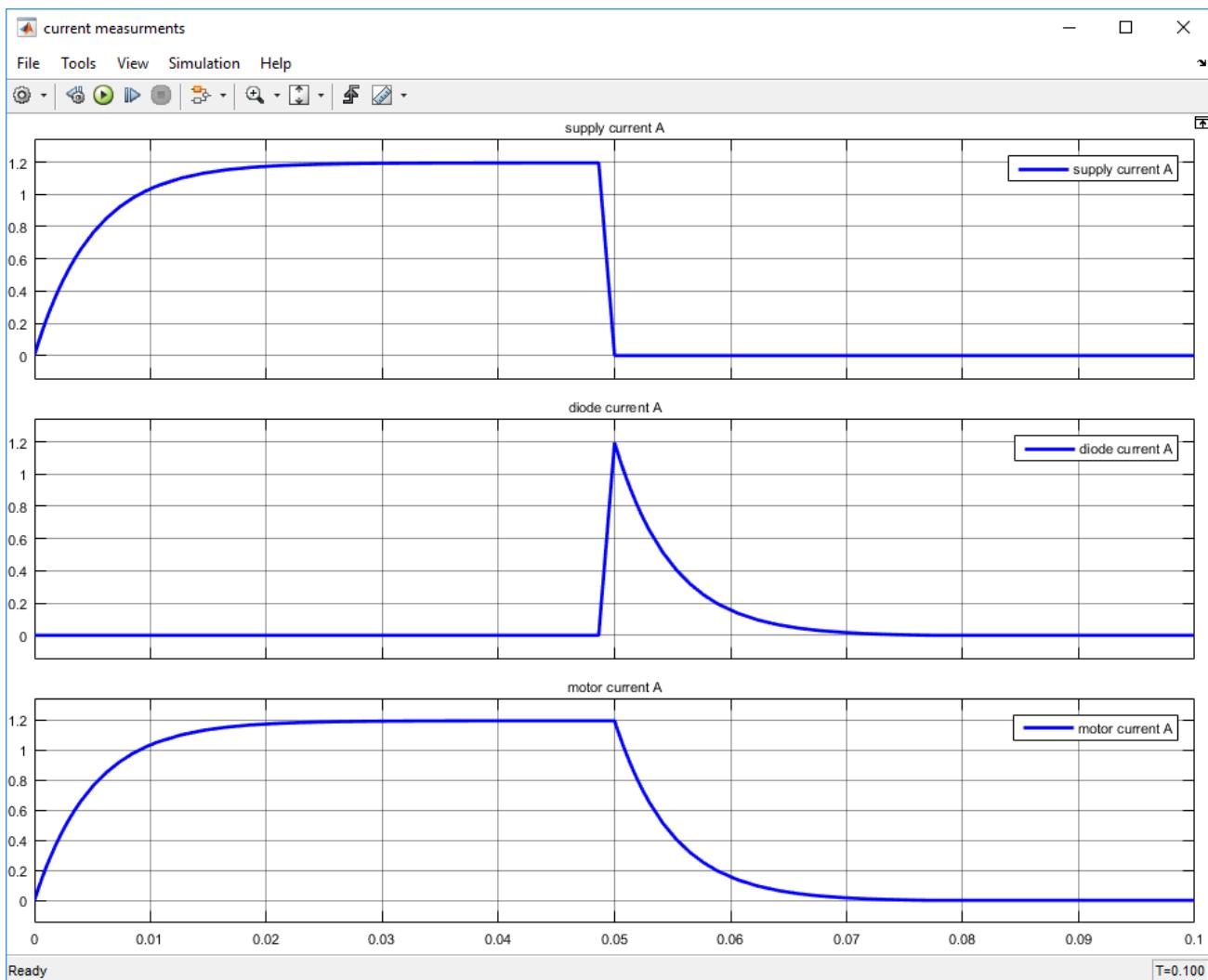


Figure 196: change in the current in the three branches of the circuit

Remember:

Objective 1: Understand the circulation of the current in the circuit in the active and freewheel phases.

The student can easily make the connection between the results obtained during simulation and the direction of the current as it passes through the various branches of the circuit. The active and freewheel phases can be identified. Objective 1 can be achieved.

3. Conclusion on rendering a model suitable for teaching

The various phases we have gone through to improve the readability of the model all contribute to making the model suitable for teaching. This is an important phase of work that enables a model that is difficult for students to read to be developed into one that can be easily understood by them. These stages require additional work from the teacher but are essential to the success of the course.

The teaching model:

- Is built to enable students to quickly understand it
- Is easily modifiable
- Enables the student to focus on the learning objective
- Must reproduce the descriptor as much as possible

The model not adapted for teaching:

- Provides the same information as the teaching model
- Has an objective that is focused on usable results
- Is difficult to read and modify
- Should not be given to students

Figure 197 shows the model not adapted for teaching and enables the potential difficulties of reading and using this model to be understood.

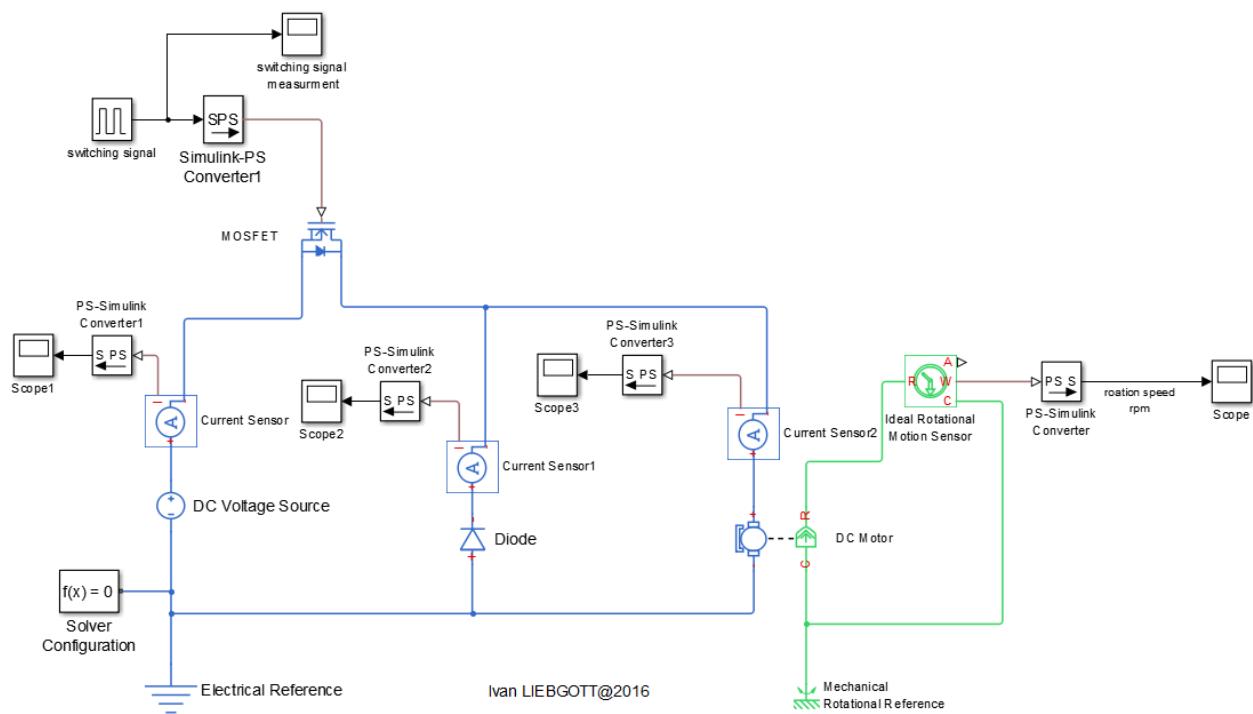


Figure 197: Model not adapted for teaching

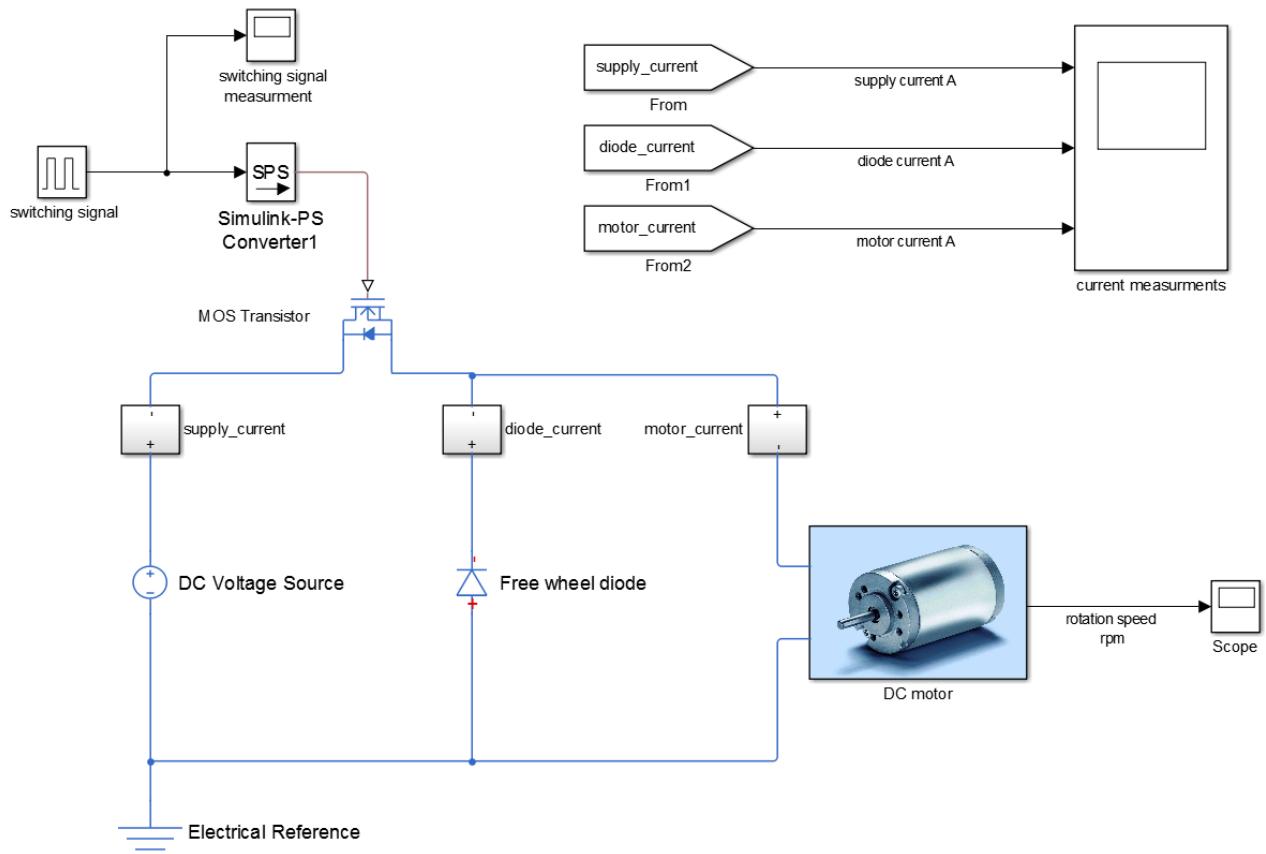


Figure 198: The teaching model

4. Optimize the instructionalization of the model based on the learning objective.

We have seen that the format of the teaching model previously established is perfectly adapted to for understanding the first objective, which was to enable students to understand the circulation of current in the buck converter in the active and freewheel phases.

In order to continue the course, and move onto the next learning objective, our model needs to be adapted further and any potential improvements that need to be made to attain later objects should be explored.

Remember:

Objectives 2, 3 and 4: Visualize and evaluate the effect of various configurations on the current ripple.

- Duty cycle
- Chopping frequency
- Load inductance value

The attention of students should be focused on the motor current for the following objectives; the two current sensors in the other branches can be removed. In addition, it is necessary to possess the average current value in order to have an image of the motor torque (Figure 199).

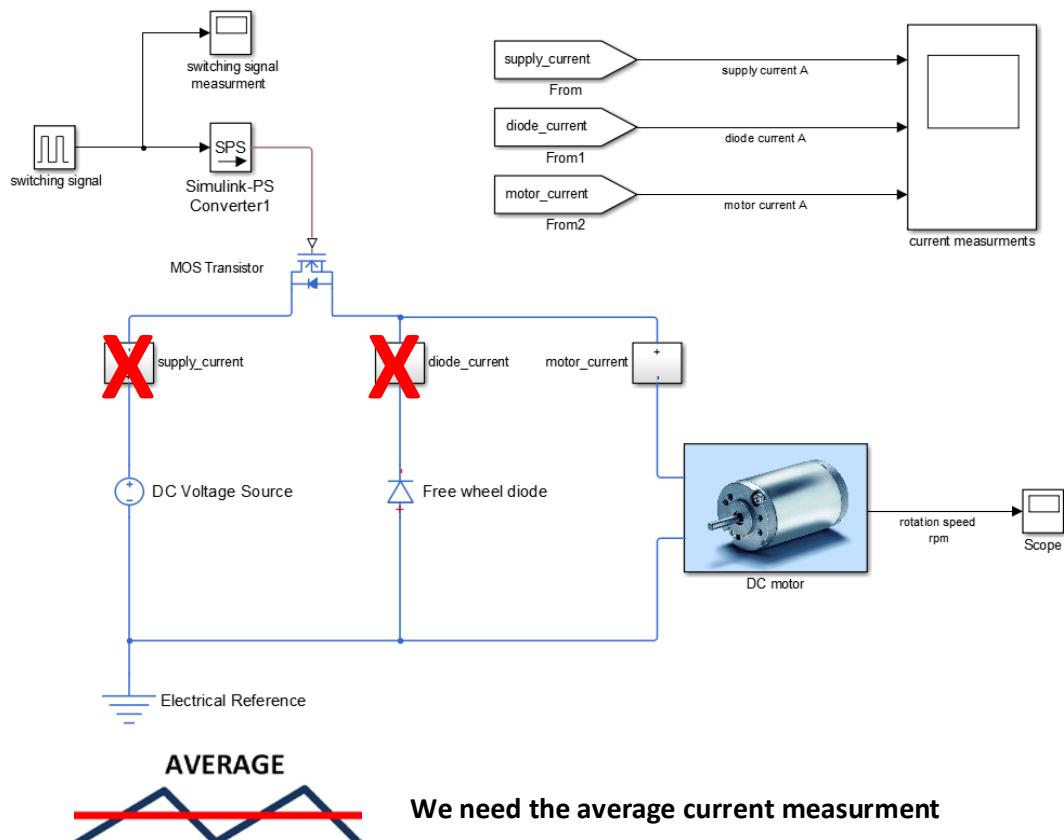
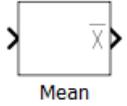


Figure 199: improving the teaching model

The **Mean** block enables the average value of a signal to be automatically calculated.

Component function	View	Library
Calculation of the average value		Simscape/SimPowerSystems/Specialized Technology/Control and Measurements Library/Measurements
Display		Simulink/Sinks

Open the “**bulk_converter_3_US.slx**” file. This file contains the teaching model with the suggested improvements. A **mask** was used on the **Mean** block to render its function more explicit (Figure 200).

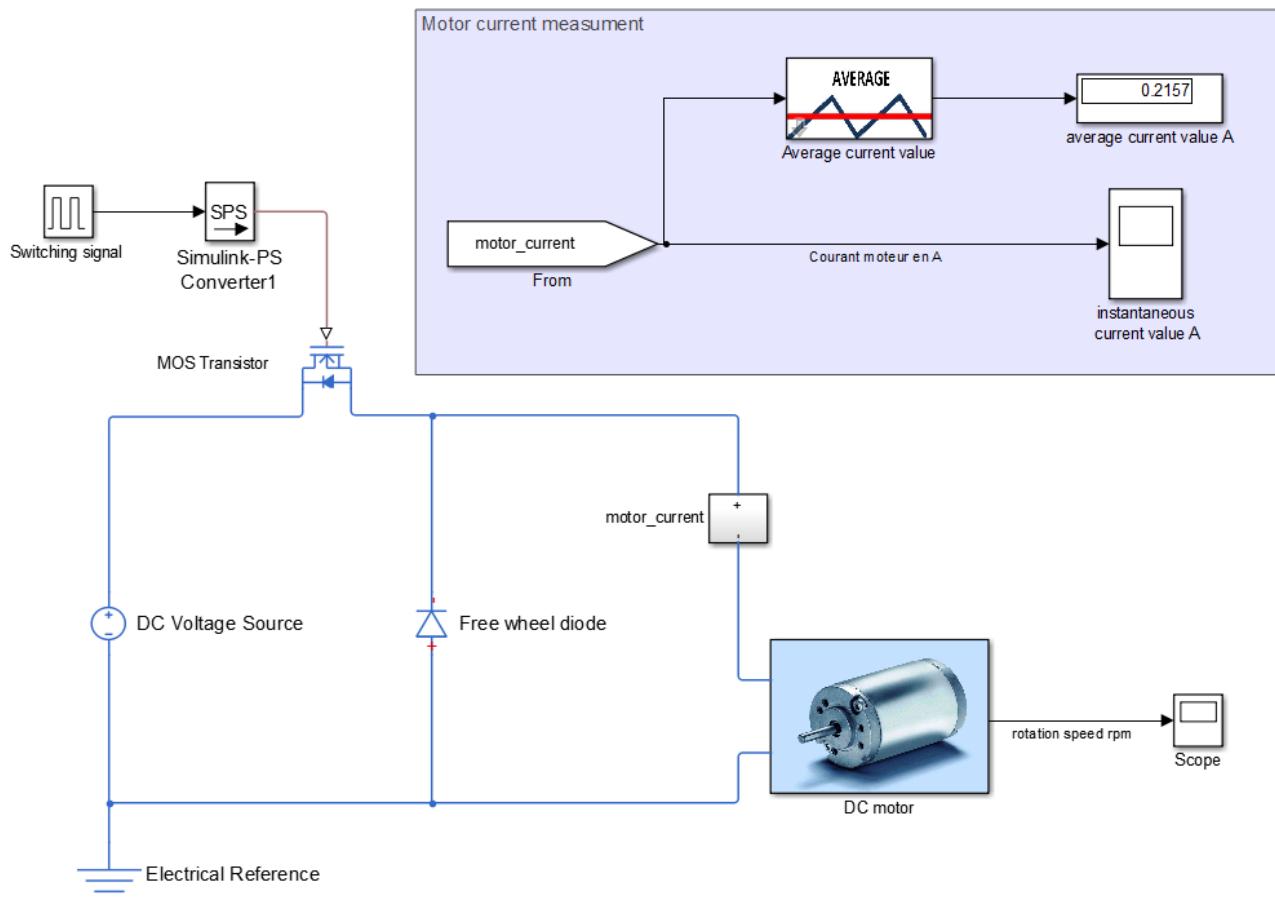


Figure 200: the optimized teaching model

Double-click the Transistor command signal block.

Adjust the **commutation period** of the transistor to **0.01s** and specify a **simulation time** of **0.1s**.

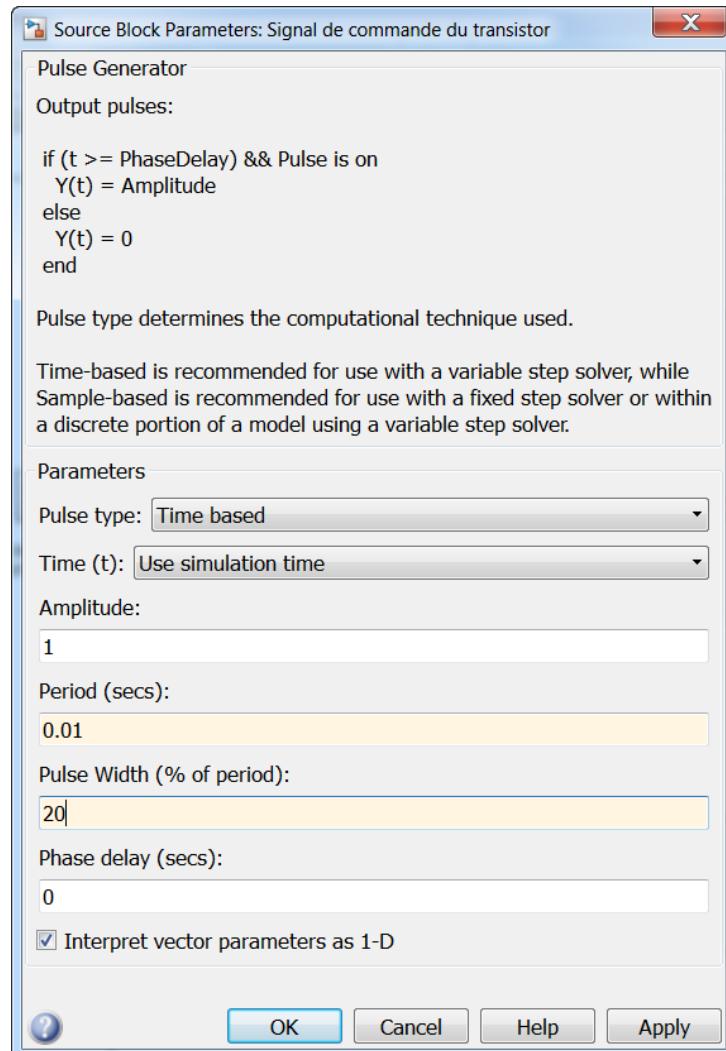
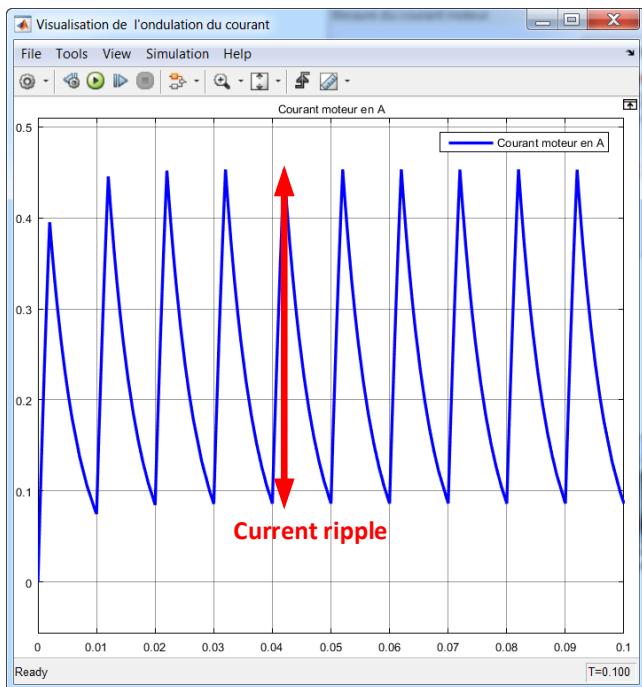
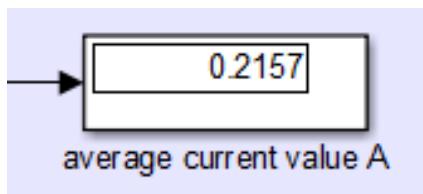


Figure 201: adjusting the commutation period of the transistor

Launch the simulation and note that the average value and appearance of the instantaneous current are not available in the model. The instantaneous value of the current is viewable in the scope, and the average value in the display unit (Figure 202).



Motor current instantaneous value and current ripple



Motor current average value

Figure 202: View of the average and instantaneous value of the motor current

We can see that the information required to reach the objectives is easily accessible for students. They will then be able to alter model parameters such as the chopping frequency (transistor commutation period) or the duty cycle.

To view the effect of the load induction, the model needs to be changed by inserting an inductance in series with the motor.

Open the “**bulk_converter_4_US.slx**” file. This file contains the previous model to which an inductance was added in series with the motor. The student will then be able to adjust the inductance value to view its effect on the motor current (Figure 203).

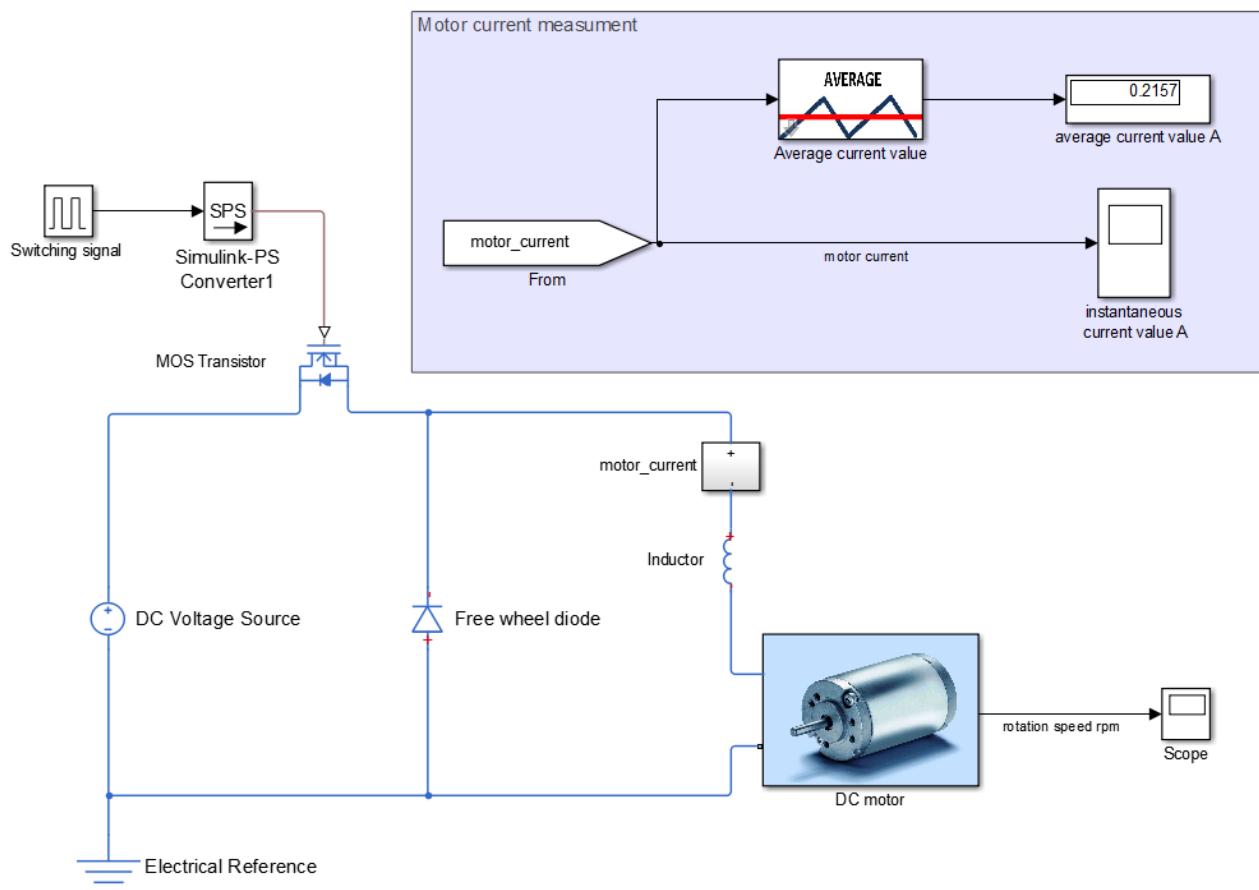


Figure 203: addition of a smoothing inductance in series with the motor

E. Using the results from the model simulation

In this section, the results from the simulation of the various models are presented and show what the student must be capable of highlighting through using the different models.

1. Objective 1: Understand the circulation of the current in the circuit in the active and freewheel phases.

The simulation of the “**bulk_converter_2_US.slx**” model (Figure 204) shows the following results:

- In the active phase, the current that passes through the motor is identical to that which passes through the supply. The current in the diode branch is zero.
- In freewheel phase, the current that passes through the motor is identical to that which passes through the diode branch. The current that passes through the supply is zero.

Circula

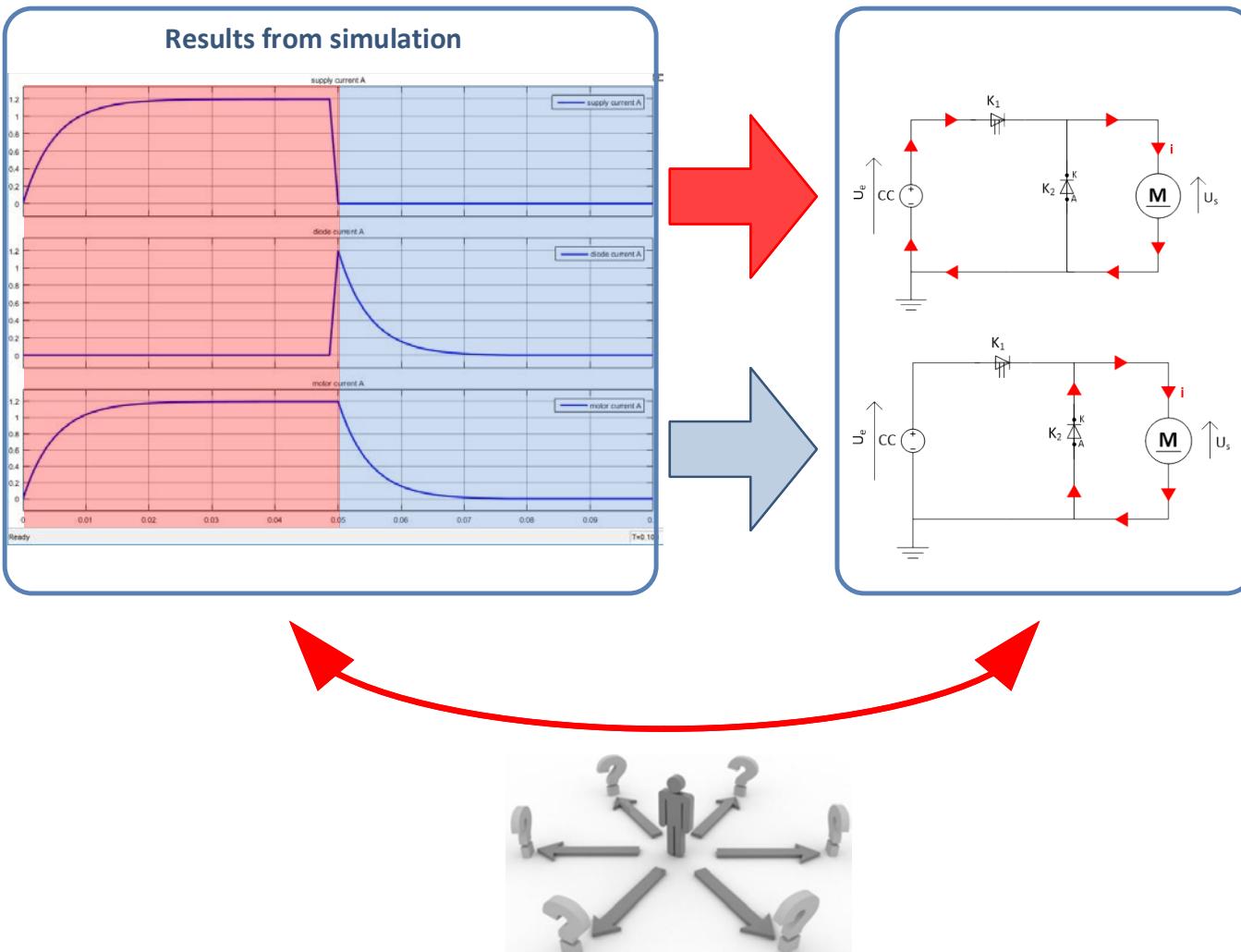


Figure 204: use and view of circulation of current in the buck converter

2. Objective 2: Visualize and evaluate the effect of the duty cycle on the motor current

The simulation of the “**bulk_converter_3_US.slx**” model (Figure 206) demonstrates the effect of the duty cycle.

Adjust the transistor’s commutation period to **0.0001s**

Adjust the Duty cycle to $\alpha = 50\%$ (Figure 205)

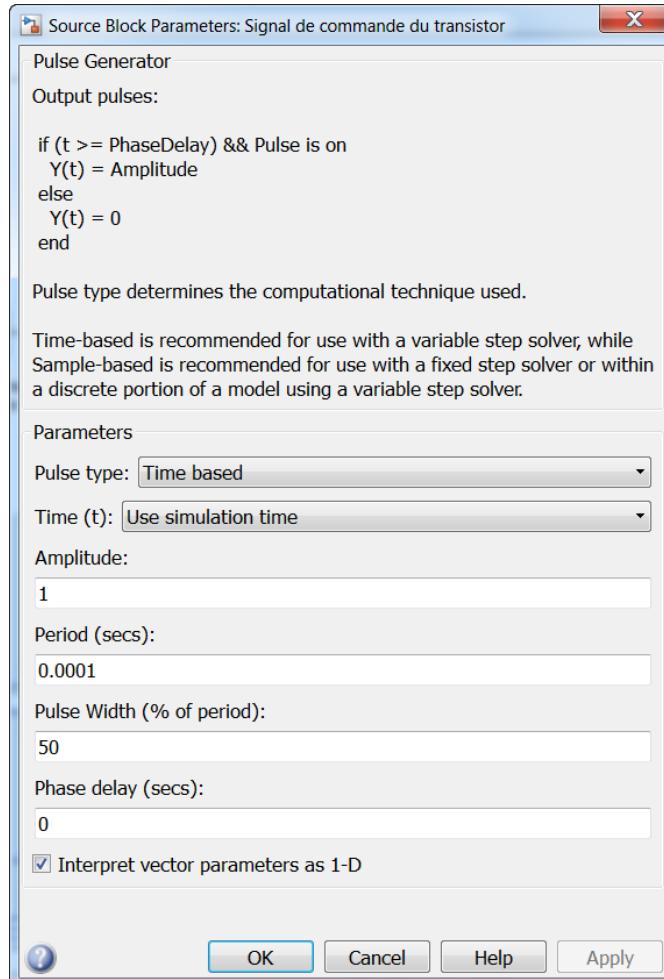
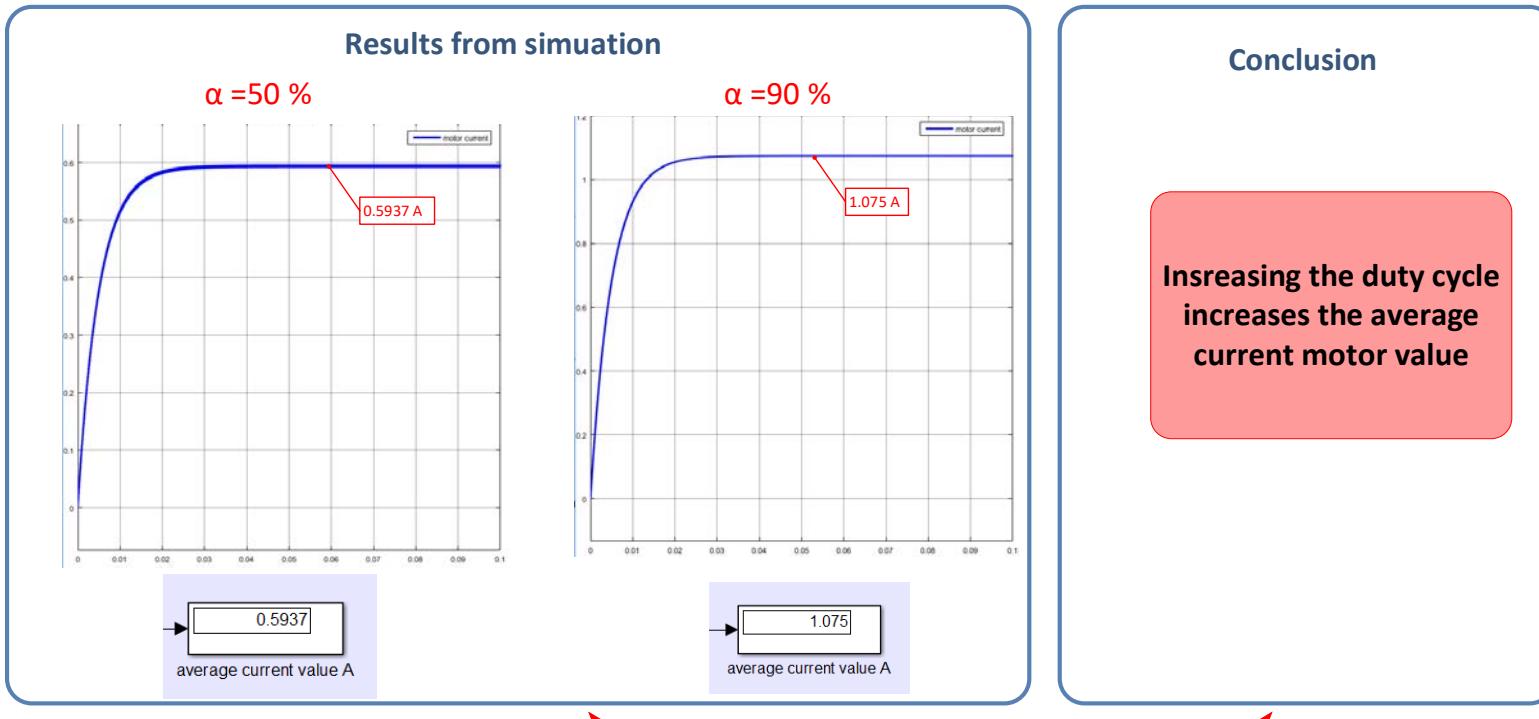


Figure 205: adjustments to the simulation parameters

Launch the simulation and increase the instantaneous speed of the motor current and its average value. Relaunch the simulation and change the value of the duty cycle to $\alpha = 90\%$.

Evaluation of the duty cycle effect



Student analysis



Figure 206: use and view of the effect of the duty cycle on the motor current

3. Objective 3: Visualize and evaluate the effect of the chopping frequency on the current ripple

The simulation of the “**bulk_converter_3_US.slx**” model (Figure 208) demonstrates the influence of the duty cycle.

Adjust the transistor’s commutation period to **0.001s**

Adjust the duty cycle to $\alpha = 50\%$

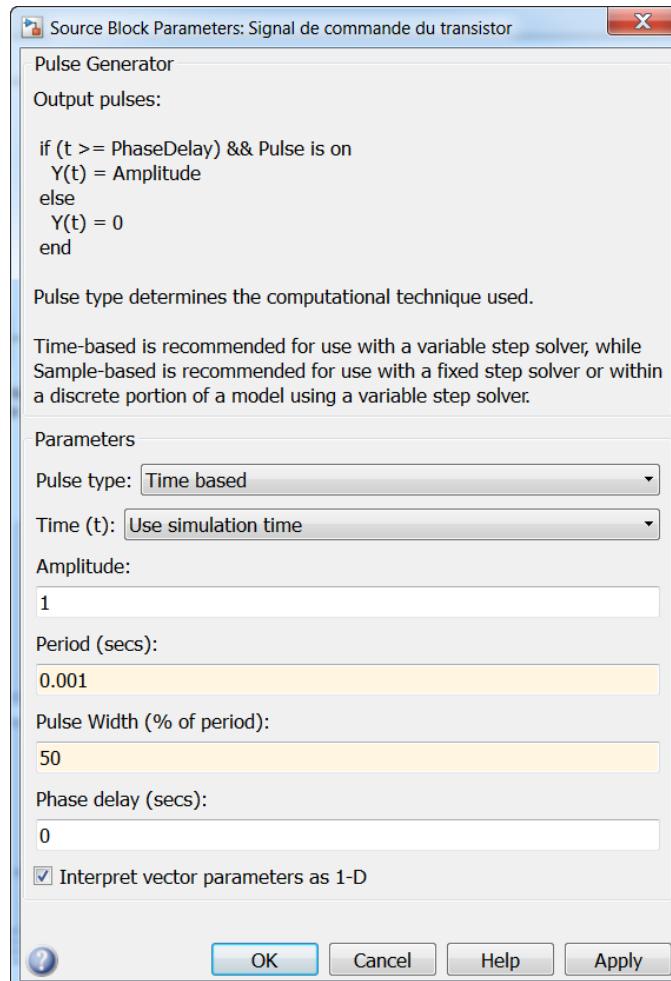
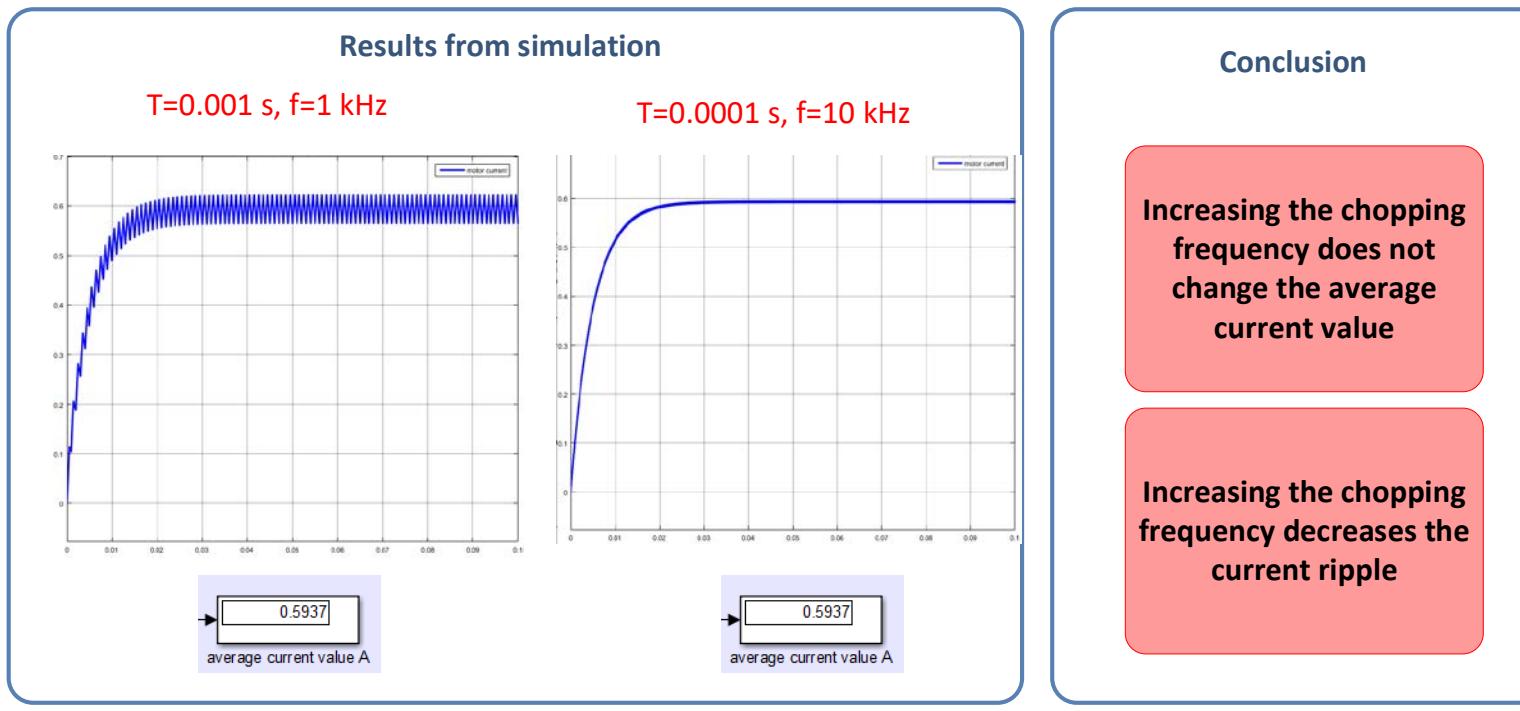


Figure 207: adjustments to the simulation parameters

Launch the simulation and increase the instantaneous motor current value and its average value.
Relaunch the simulation and change commutation period to **T=0.0001 sec**

Evaluation of the chopping frequency effect



Student analysis



Figure 208: use and visualization of the effect of the chopping frequency on the motor current

4. Objective 4: Visualize and evaluate the effect of the load inductance on the current ripple

The simulation of the “**bulk_converter_4_US.slx**” model (Figure 210) demonstrates the influence of the load inductance.

Adjust the added inductance value to **1 mH**

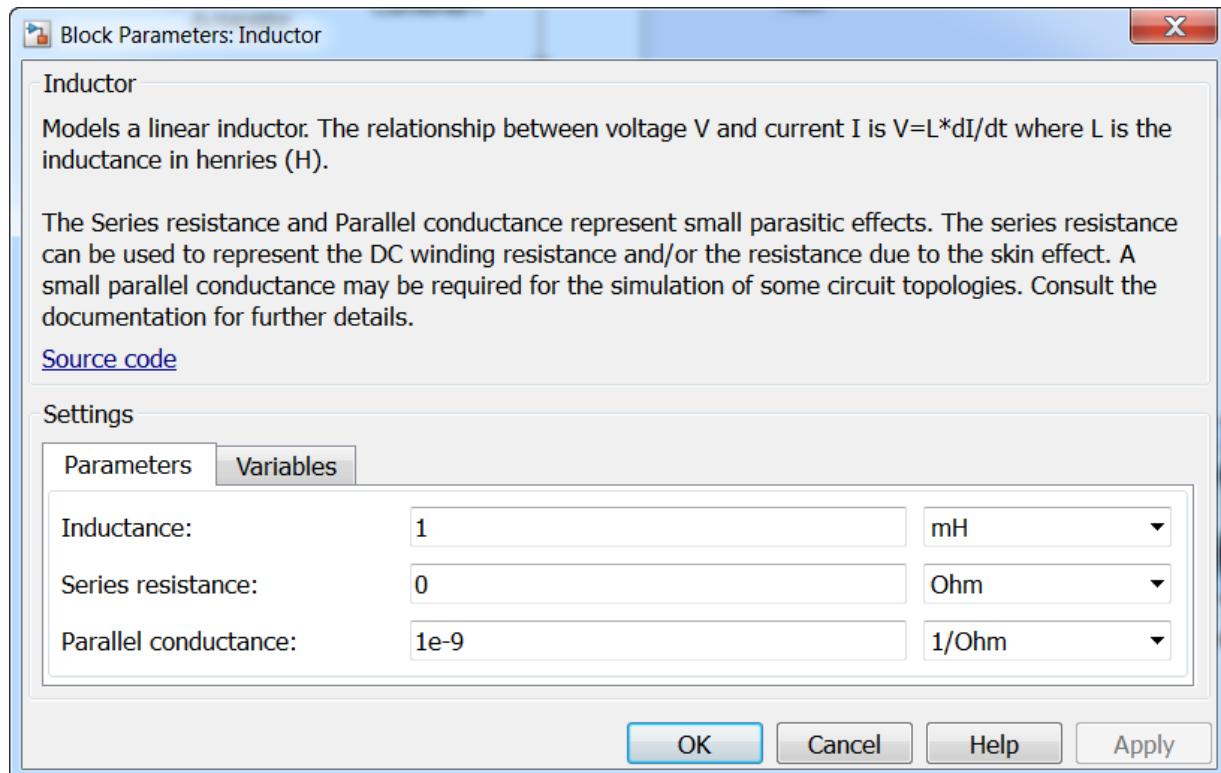


Figure 209: adjustments to the simulation parameters

Launch the simulation and increase the instantaneous speed of the motor current and its average value. **Relaunch** the simulation by changing the inductance value to **1 H**.

Evaluation of the inductance effect

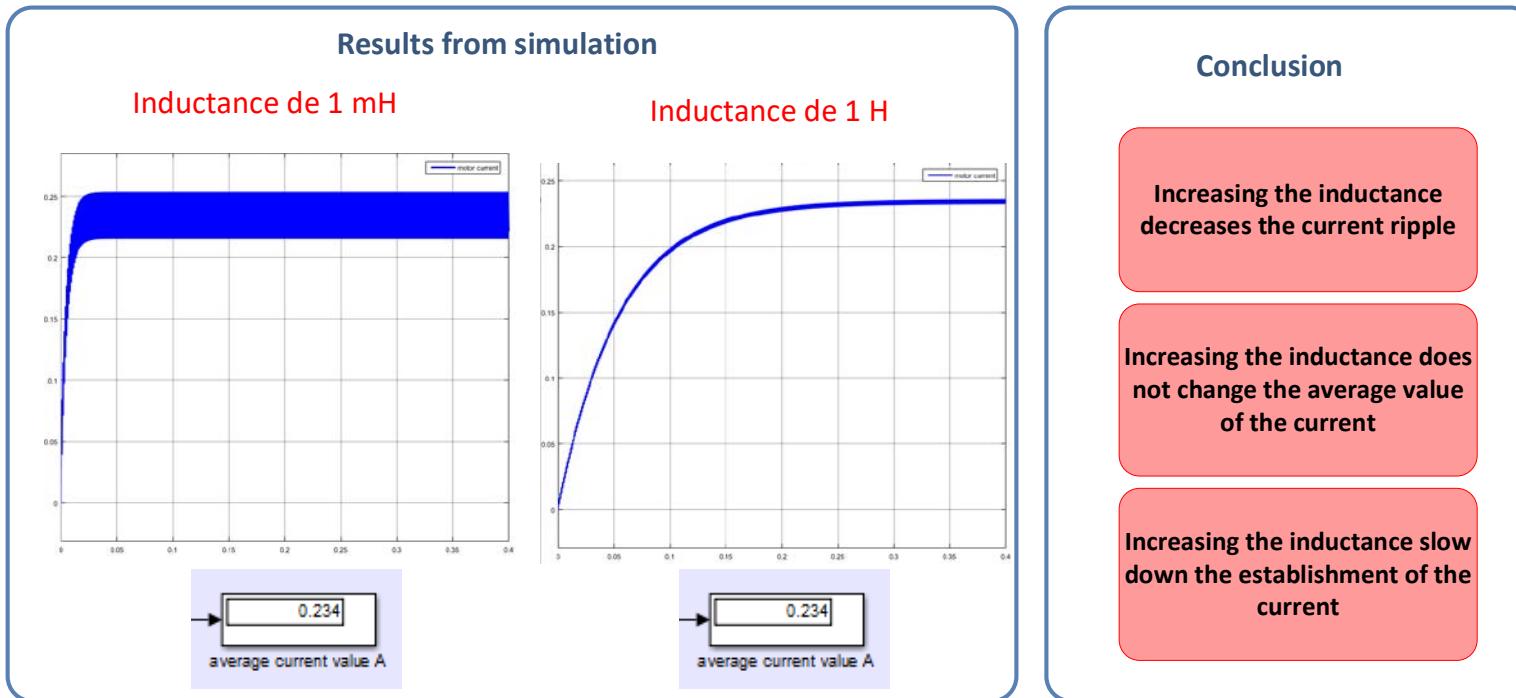


Figure 210: View and use of the effect of load inductance on the motor current

F. Conclusion

The success of a learning course is not limited to building models and their simulation. The model must be made suitable for teaching and learning. These changes must enable the form of the model to be optimized for the learning objective. Students are then in the best position to focus their attention on achieving the learning objective.

Chapter 4: MATLAB management

I. Introduction

The aim of this chapter is to present the basic functionalities of **MATLAB** and the most use commands to begin with. The **MATLAB** calculation possibilities are vast, and here we present basic commands that can be useful as for Engineering Sciences calculations.

All of the following commands will be entered into the **MATLAB** “command window”. Each line of commands must be validated by the “enter” key on the keyboard to be processed by the software.

A. Creating a variable

Creating variable a:

```
>> a=1 (validate)  
a =  
1
```

MATLAB differentiates between capital and lower case letters: **a** and **A** will be two distinct variables.

Note that MATLAB responds that it just created variable **a**. This new variable appears in the **Workspace** window and the command that we just entered appears in the **Command History** window.

Creating variable b:

```
>> b=5;
```

The semicolon at the end of the command line indicates that we do not require a response to our command. Variable b appears in the **Workspace**.

It is easy to perform an operation between two variables, for example, multiplication.

```
>> a*b  
years =  
5
```

All of the commands from the **Command History** window can be easily recalled by pressing the up and down keys on the keyboard.

B. Creating a vector

Let's consider the series of data in the table shown in Figure 211.

x	0	10	20	30	40	50
y1	0	16	35	49	25	6
y2	0	12	21	36	49	56

Figure 211: table of data

If we wish to use this data, the need to be entered into MATLAB in the form of vectors.

The separator can be a space:

```
>> x=[0 10 20 30 40 50];
```

The separator can be a semicolon:

```
>> y1=[0,16,35,49,25,6];
>> x=[0 12 21 36 49 56];
```

Each component *i* of vector *x* is stored in variable *x(i)*.

```
>> x(2)
years =
10
```

MATLAB also offers the automatic vector creation functions.

```
>>u=[8:0.1:50]
```

This command enables the creation of vector **u** that will have all of the values between **8** and **50** with steps of **0.1** as its components. Using this command enables the spacing between components to be specified, with the number of components being calculated based on the specified steps and limits.

If the step is not specified, it is assigned the value of 1 by default.

```
>>u=[8 :50]
```

This command enables the creation of vector **u** that will have all of the values between **8** and **50** with steps of 1 as its components.

If on the contrary we wish to specify the exact number of components, the *linspace* function can be used.

```
>>u=linspace(8,50,1000)
```

This command enables the creation of vector **u** that will have 1000 components regularly spaced between the limits of 8 and 50.

C. Indexing vector components

The components of a vector can be easily accessed by using line indexing.

Variable **u(i)** contains the value of the *i*th component of vector **u**.

```
>>u(56)
years=
10.313
```

D. Plotting curves

Two vectors of the same length can be plotted using the *plot* command:

```
>> plot(x,y1)                                trace vector y1 as a function of vector x
```

A graphics window “Figure 1” opens with the requested curve.

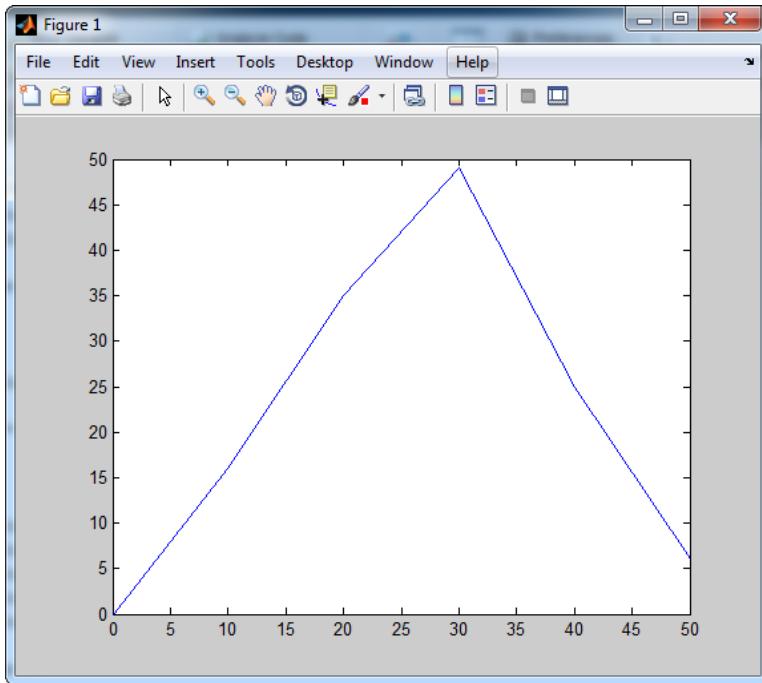


Figure 212: plot of two vectors

We can plot vector y_2 as a function of vector x

```
>> plot(x,y2)                                plot vector y2 as a function of vector x
```

We notice that in the window “Figure 1” only curve y_2 as a function of x is displayed. The preceding curve was deleted.

We can use the *hold on* command to save the curves in the same window.

```
>> hold on  
>> plot(x,y1)
```

The *hold off* command cancels the *hold on* command.

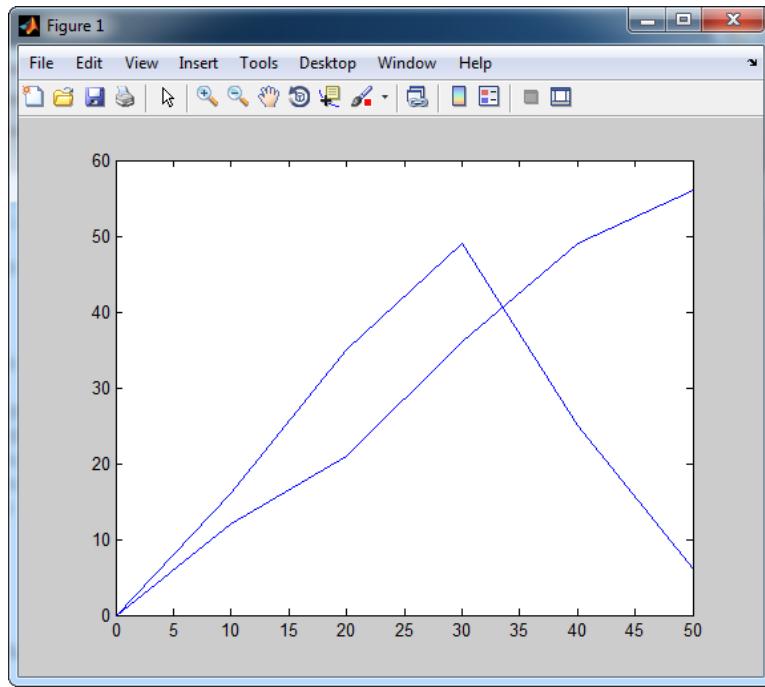


Figure 213: plot of two curves in the same graphics window

The *grid on* command displays the grid

```
>>grid on
```

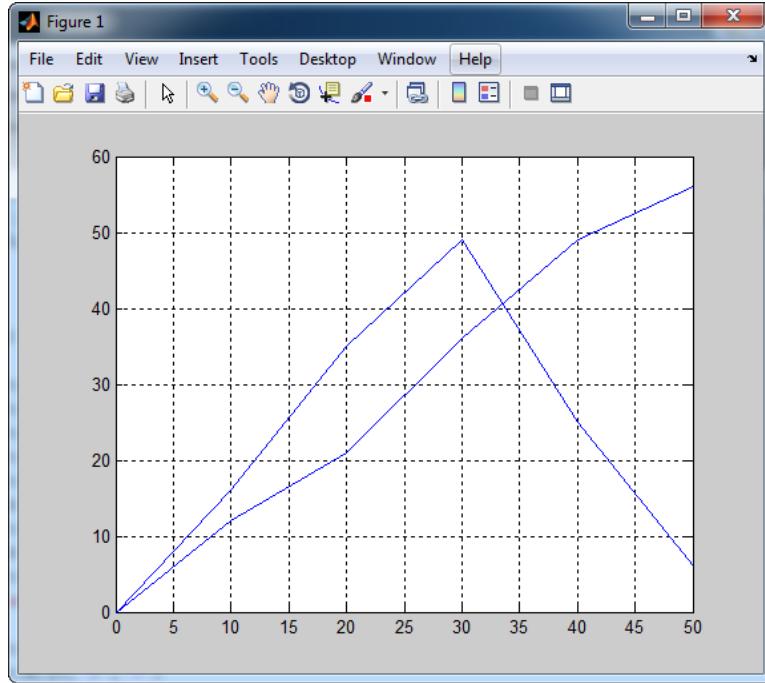


Figure 214: displaying the grid on a graphic

The *grid off* command hides the grid.

If we want to open a new graphics window, we need to use the *figure* command

```
>> figure
```

An empty graphics window named figure2 opens and it is possible to work in this new window (Figure 215).

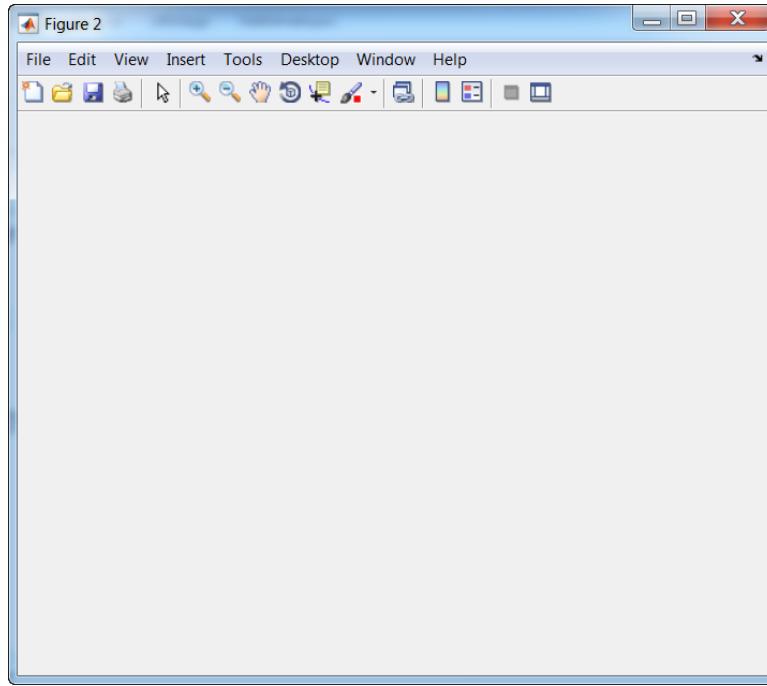


Figure 215: opening a new graphics window

E. Elementary curve shaping

Functions for shaping curves are also available. The *plot* function accepts a third optional argument that enables you to specify shapes for curves.

The table in Figure 216 presents the codes that can be entered to obtain shapes for curves. These codes can be combined in any order.

Color	Marker	Line style
b Blue	- Point	- Continuous
g Green	o Circle	-- Hyphen
r Red	x Diagonal cross	: Dotted line
c Cyan	+ Vertical cross	-. Line-point
m Magenta	*	
y Yellow	s Square	Note: If a marker style is specified without any line style, no line will be drawn.
k Black	d Diamond	
w White	v Downward	
	^ triangle	
	< Upward triangle	
	> Left triangle	
	p Right triangle	
	h Pentagram	
	Hexagram	

Figure 216: codes for shaping curves

You can experiment:

```
>>hold off  
>>plot(x,y1,'r*:')
```

r for red, * for markers and : for the dotted line style

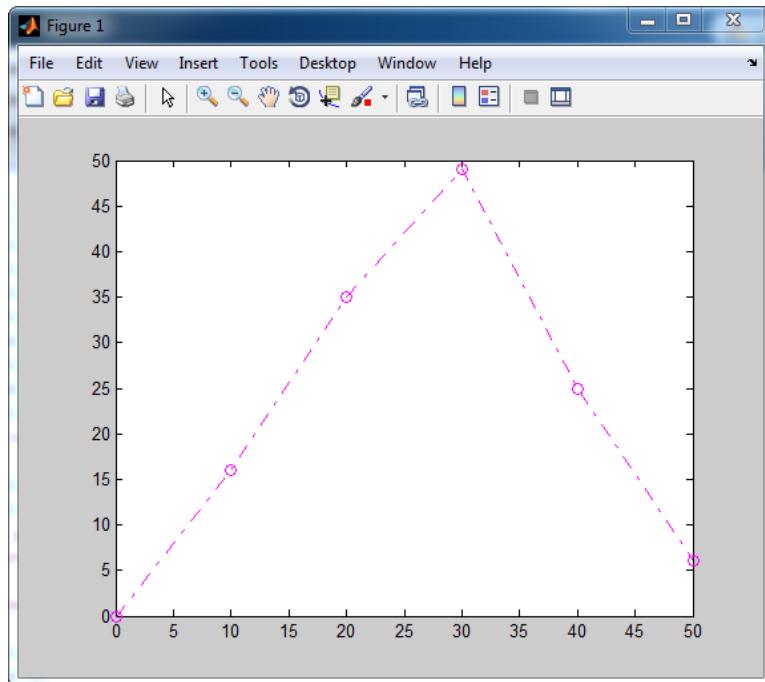


Figure 217: shaping a curve: modifying line color and style

```
>>plot(x,y1,'mp')
```

No line style is specified, only the markers appear.

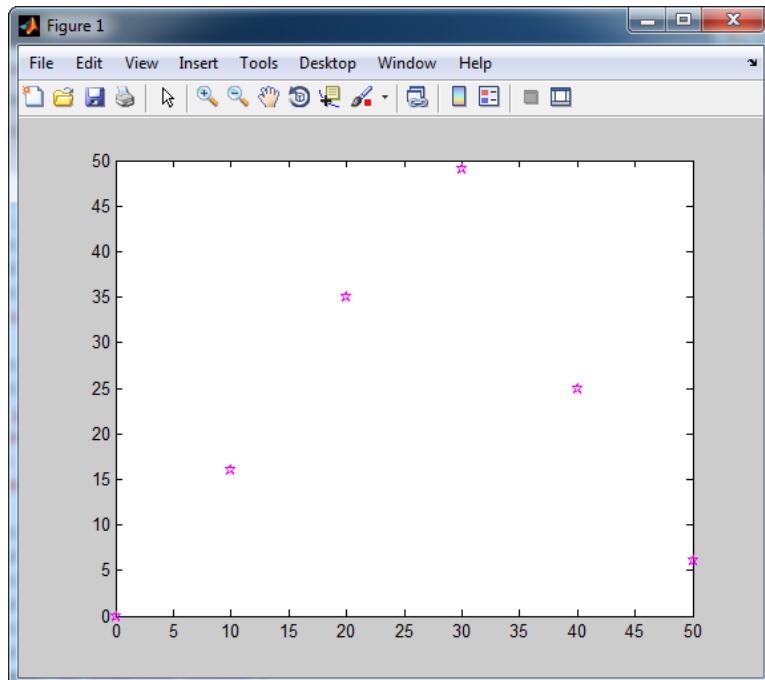


Figure 218: shaping a curve: adding markers

It is also possible to change the thickness of the line with the *LineWidth* argument

```
>> plot(x,y1,'mo-.','LineWidth',3)
```

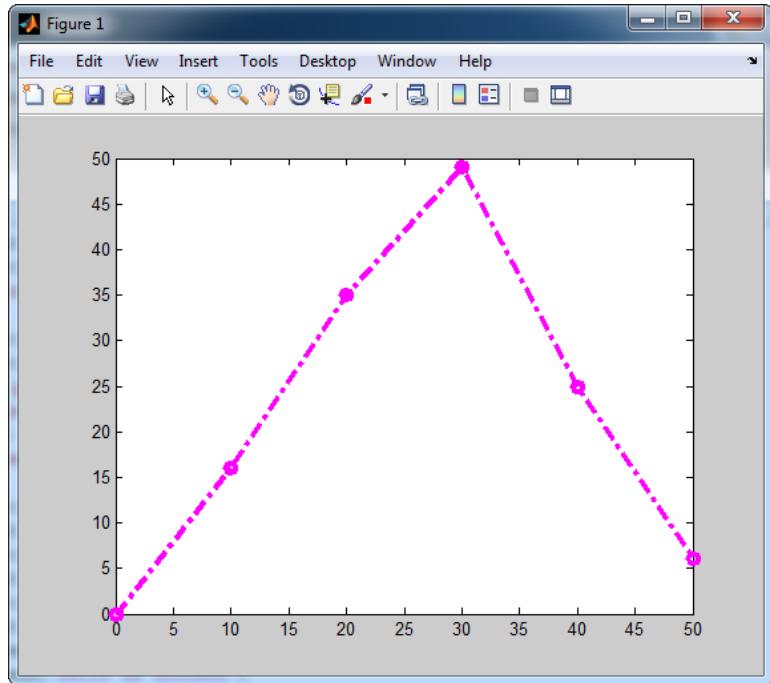


Figure 219: shaping a curve: choosing line thickness

F. Graphics annotation

It is also possible to annotate the graphic:

```
>> title('my first curve')          displays a title on the graphic  
>> xlabel('vector x')              abscissa axis label  
>> ylabel('vector y1')            ordinate axis label  
>> legend('y1 as a function of x') adding a legend
```

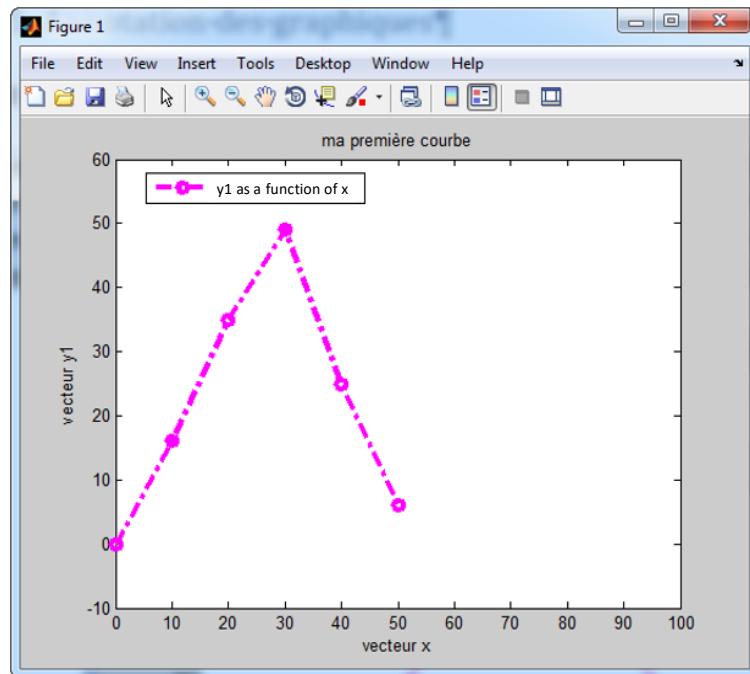


Figure 220: shaping a curve: legend and axis annotation

The `axis` command enables the axis scales to be manually defined.

```
axis([0 100 -10 60])
```

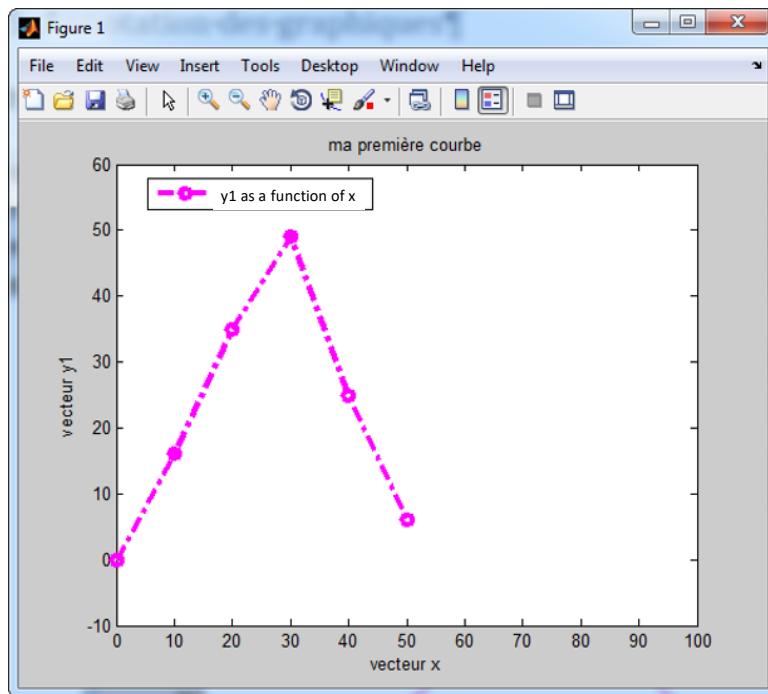


Figure 221: shaping a curve: modifying axis scales

G. Creating an elementary script

A script is a program written in **MATLAB** language. Understandable, it is very useful to group a series of command lines in a file called a script. It offers the advantage of being able to save and run the same sequence numerous times.

To create a script, click **New script**



in the command bar.

An **Editor** window appears where command lines can be entered.

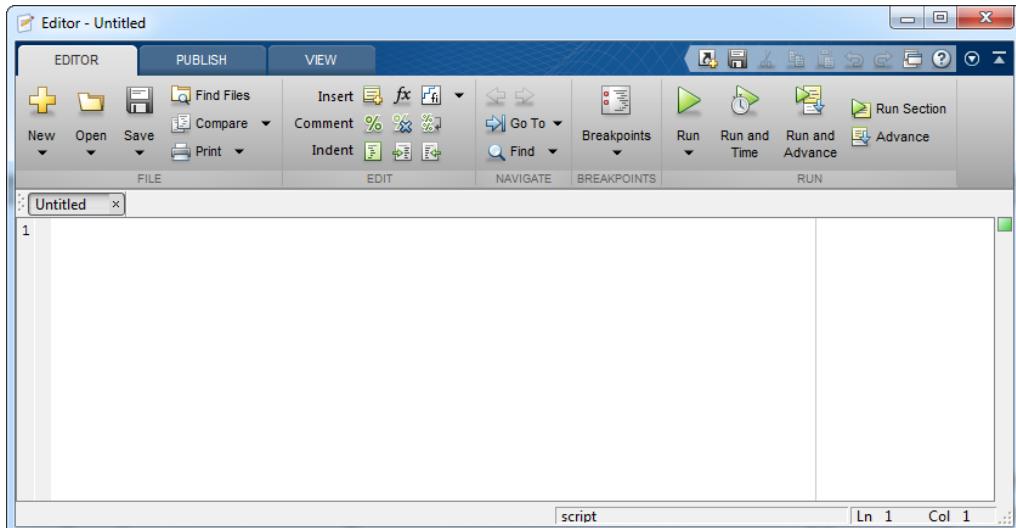


Figure 222: Editor window for editing scripts

Enter the command lines and save the file under the name **plot_sinus_0.m** (the extension **.m** is the extension for MATLAB script files, and the extension **.m** will be issued by default)).

The **%** sign indicates a comment in a script.

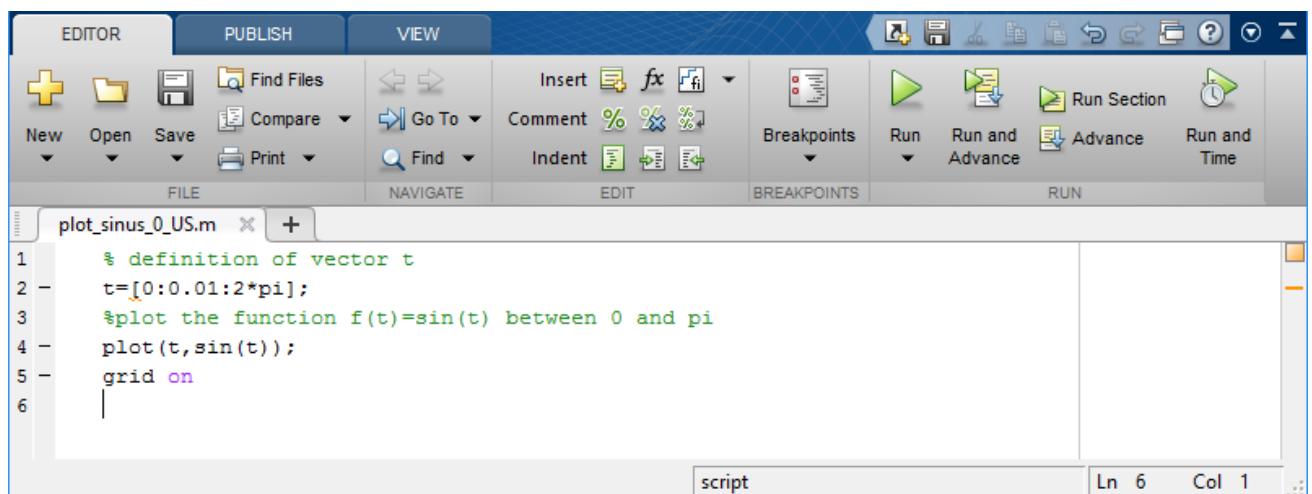


Figure 223: first script with MATLAB

Run the script by clicking



on the command bar.

If the window from Figure 224 appears, the file location does not belong to the MATLAB “path” and MATLAB cannot execute the script. The software offers you the options of:

- Adding the folder to the “path”: **Add to Path** (recommended)
- Moving this file to the current **MATLAB** work folder: **Change FOLDER**

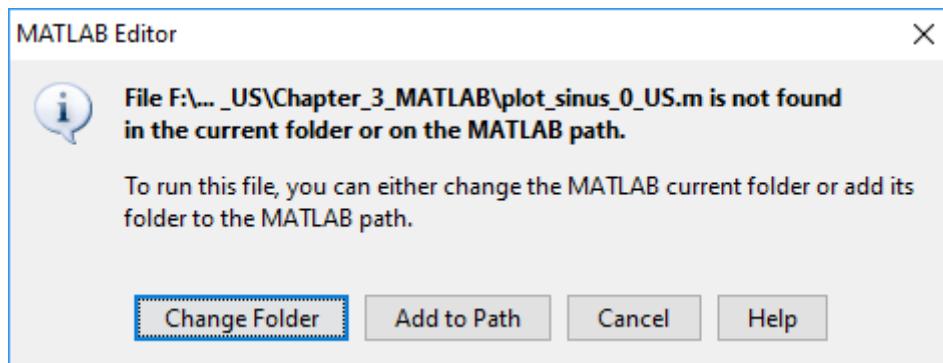
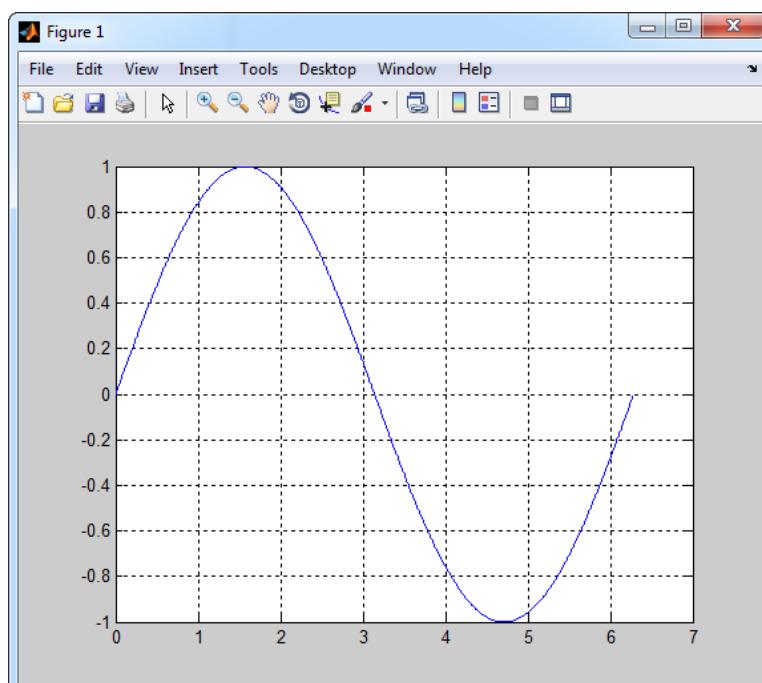
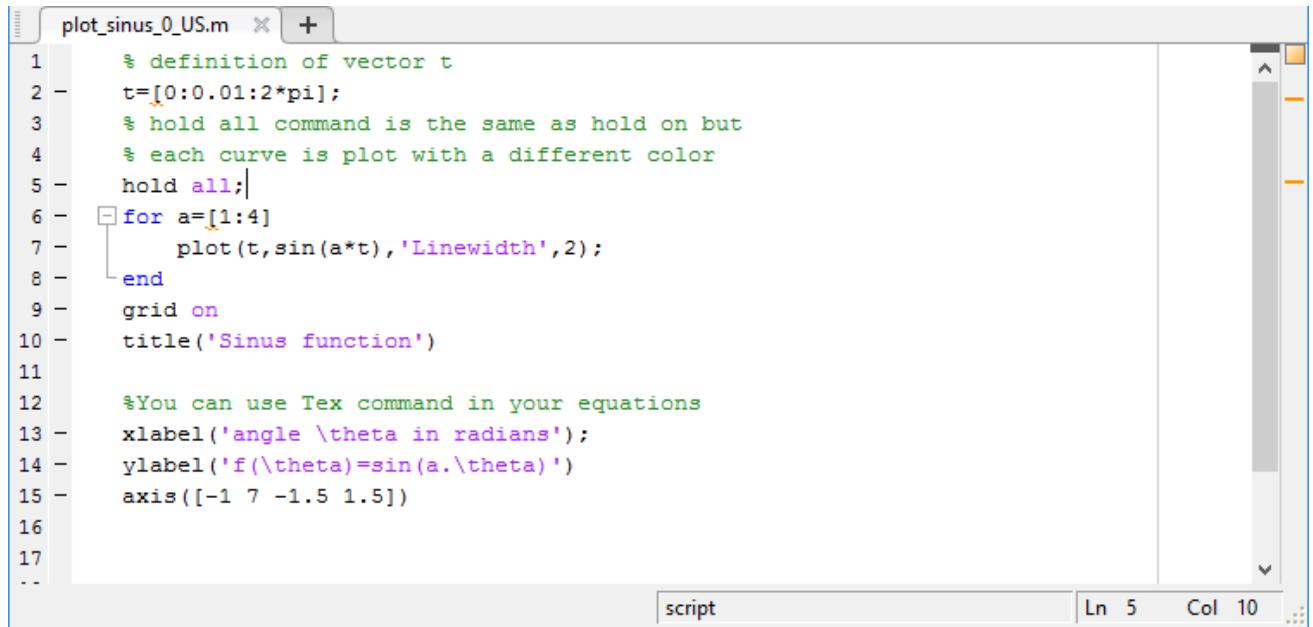


Figure 224: window showing automatic addition of the folder to the MATLAB “path”

We obtain the expression of the sine function as a function of time.



The classic programming commands can be used as a **for** loop. Modify your script as indicated in Figure 225.



```

plot_sinus_0_US.m
1 % definition of vector t
2 t=[0:0.01:2*pi];
3 % hold all command is the same as hold on but
4 % each curve is plot with a different color
5 hold all;
6 for a=[1:4]
7 plot(t,sin(a*t),'Linewidth',2);
8 end
9 grid on
10 title('Sinus function')
11
12 %You can use Tex command in your equations
13 xlabel('angle \theta in radians');
14 ylabel('f(\theta)=sin(a.\theta)')
15 axis([-1 7 -1.5 1.5])
16
17

```

Figure 225: Several sines plotted on the same graphic

Run the script and view the result.

If a syntax error is detected whilst running the script, the details will appear in red in the command window.

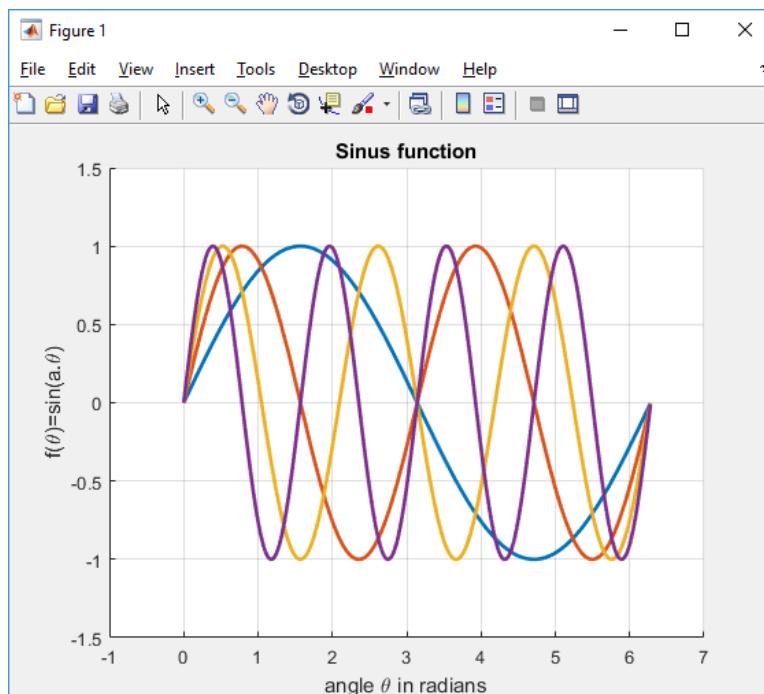


Figure 226: results of the script for plotting several sines

H. MATLAB comparison operators

MATLAB possesses comparison operators and logical operators to carry out tests in scripts.

Useful commands	
Operators	MATLAB Syntax
Equal to	$=$
Different from	\neq
Greater than	$>$
Greater than or equal to	\geq
Less than	$<$
Less than or equal to	\leq
or	$ $
and	$\&$

Figure 227: MATLAB logic and comparison operators

I. Structure of normal loops

In order to implement loops in scripts, the three most common solutions are:

- if – elseif – else
- for
- while

1. Syntax for the if – elseif – else loop

```
if (logic test 1)
    action 1
elseif (logical test 2)
    action 2
else
    action 3
```

For this type of loop, a single action among *action1*, *action 2* and *action 3* will be carried out. The tests will be performed in order and only if this is necessary. Thus if (*logical test 1*) is true, then *action 1* will be run and (*logical test 2*) will not be evaluated. There may be any number of *elseif* instructions or none. There may only be a single *else* instruction that must represent the final condition.

2. Loop syntax for

```
for variable=value:initial:increment:value_final
    action
end
```

The *for* loop runs the action a defined number of times by changing the value of one variable.

3. Syntax of the *while* loop

```
while condition  
    action  
end
```

The action is carried out provided that the condition evaluated is true. The exit from the loop proceeds when the condition evaluated becomes false.

II. Example of uses

A. Interpolating a series of data

When analyzing experimental data, it is often necessary to interpolate a series of data. MATLAB possesses functionalities that enable interpolation to be simply carried out.

Let's consider a series of experimental points represented by two vectors $X=[0\ 10\ 20\ 30\ 40]$ and $Y=[2\ 5\ 12\ 25\ 46]$.

The objective is to place these points on a graphic and plot the interpolation curve.

In the command window, **create** the two vectors X and Y, and view their distribution by plotting the experimental points obtained.

```
>>X=[0 10 20 30 40];  
>>Y=[2 5 12 25 46];  
>>plot(X,Y,'r.','MarkerSize',25); specifying the marker without specifying the line style displays only the markers
```

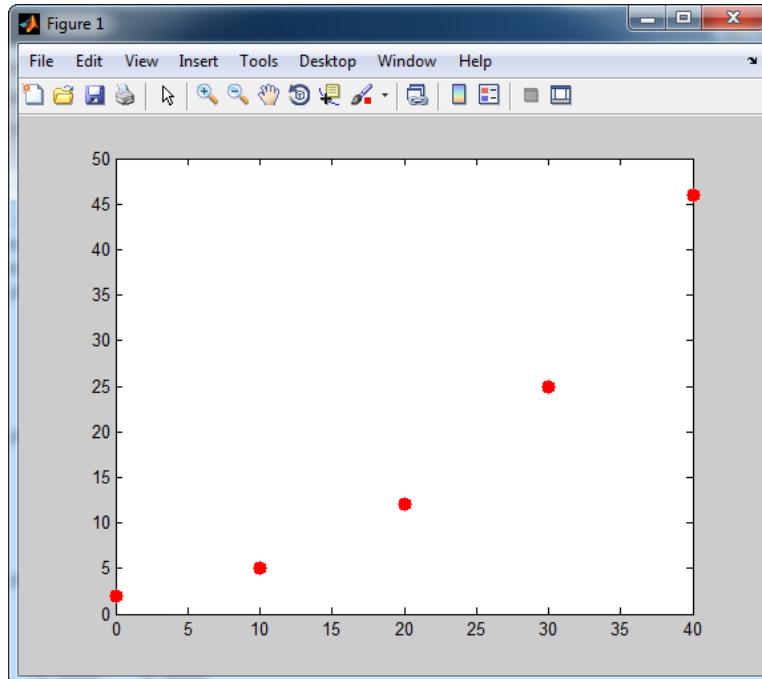


Figure 228: distribution of a series of points

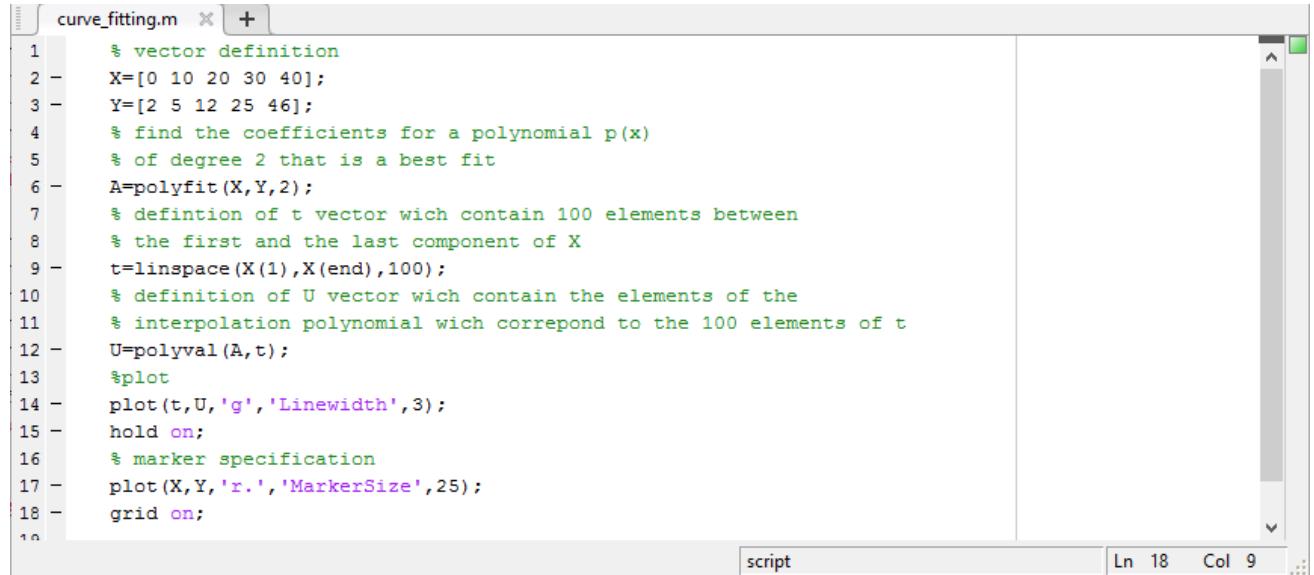
The appearance of the distribution enables us to search for an interpolation with a second degree polynomial. The *polyfit* function automatically gives the coefficients of the interpolation polynomial.

```
>> polyfit(X,Y,2)
```

```
years =
0.0300 -0.1200 2.4000
```

The second degree interpolation polynomial expression is $p(x) = 0.03x^2 - 0.12x + 2.4$.

Test for the script of Figure 229 to understand the value using a script to automate this type of task. This script enables various points (round red) to be placed and the interpolation curve to be superimposed.



```
curve_fitting.m
1 % vector definition
2 X=[0 10 20 30 40];
3 Y=[2 5 12 25 46];
4 % find the coefficients for a polynomial p(x)
5 % of degree 2 that is a best fit
6 A=polyfit(X,Y,2);
7 % defintion of t vector wich contain 100 elements between
8 % the first and the last component of X
9 t=linspace(X(1),X(end),100);
10 % definition of U vector wich contain the elements of the
11 % interpolation polynomial which correspond to the 100 elements of t
12 U=polyval(A,t);
13 %plot
14 plot(t,U,'g','LineWidth',3);
15 hold on;
16 % marker specification
17 plot(X,Y,'r.','MarkerSize',25);
18 grid on;
```

Figure 229: interpolation of a series of data

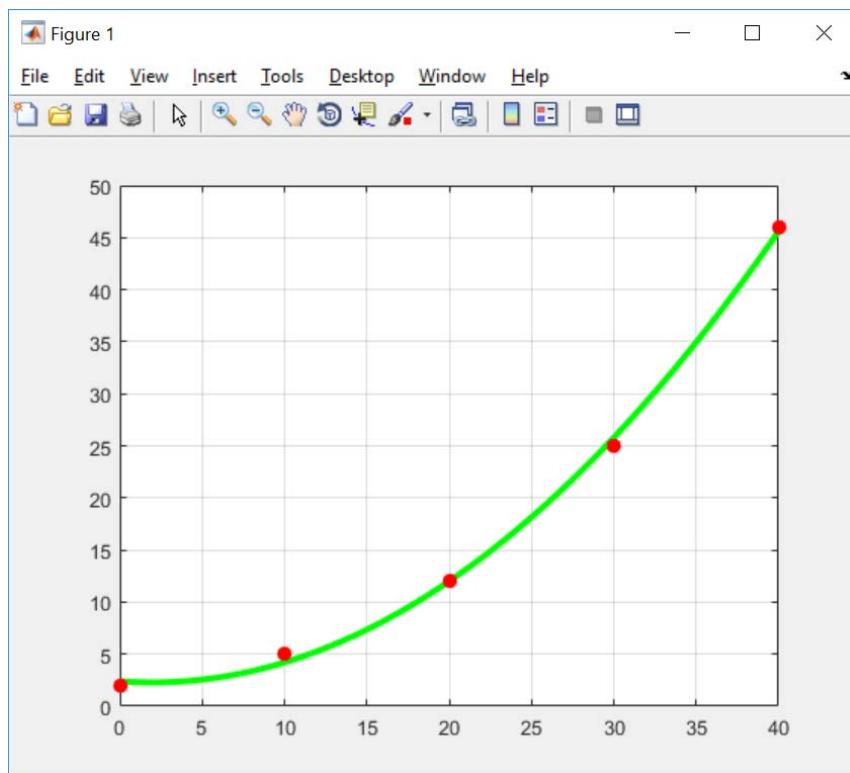


Figure 230: graphic showing the interpolation of a series of data

B. Symbolic calculation with MATLAB

In the modelling phase, the student and engineer often need to resolve algebraic or differential equations. In this section we will see how to carry out a calculation using symbolic variables, solve equations, and work with Laplace transforms.

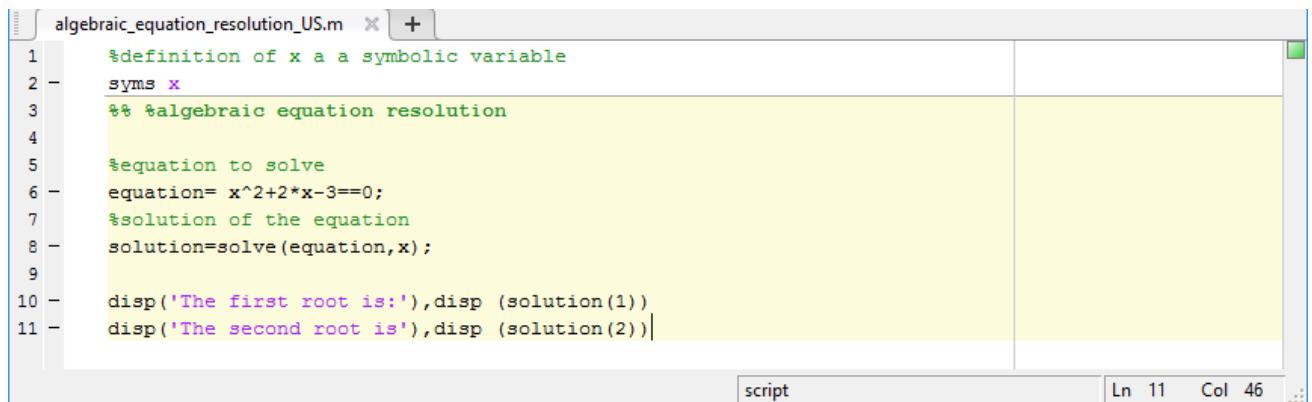
1. Solving an algebraic equation

Consider the following problem: $x^2 + 2x - 3 = 0$

We want to find the values of x that verify this equation by creating a script.

We use the `syms` command which allows MATLAB to treat a variable as a symbolic variable.

Input the following script or open the script “**algebraic_equation_resolution_US.m**”:



```
algebraic_equation_resolution_US.m
1 %definition of x as a symbolic variable
2 syms x
3 %% Algebraic equation resolution
4
5 %equation to solve
6 equation= x^2+2*x-3==0;
7 %solution of the equation
8 solution=solve(equation,x);
9
10 disp('The first root is:'),disp (solution(1))
11 disp('The second root is'),disp (solution(2))
```

Figure 231: script for solving a second degree equation

Executing the script provides the following result:

```
>>algebraic_equation_resolution
```

The first root is:

-3

The second root is:

1

MATLAB returns the values for the two roots of this equation.

The same method is used to solve equations with a complex root. Consider the following problem:

Consider the following problem: $x^2 + 2x + 5 = 0$

Modify your script or open the script “**algebraic_equation_resolution_complex_root_US.m**”:

```

expanding_factoring_expression_US.m    algebraic_equation_resolution_complex_root_US.m    +
1 %definition of x as a symbolic variable
2 syms x
3 %% resolution of an algebraic equation (complex roots)
4
5 %equation to solve
6 equation= x^2+2*x+5==0;
7 %Resolution
8 solution=solve(equation,x);
9
10 disp('The first root is:'),disp (solution(1))
11 disp('The second root is:'),disp (solution(2))
12

```

Ln 1 Col 1

Figure 232: script for solving a second degree equation with complex roots

Executing the script provides the following result:

```
>> algebraic_equation_resolution_complex_root_US
```

The first root is:

```
- 1 - 2i
```

The second root is:

```
- 1 + 2i
```

MATLAB returns the values for the two roots for this equation in the form of a complex number.

2. Expanding or factoring an expression

It is often useful to change the form of an expression by factoring it or expanding it. In order to use the *expand* and *factor* functions, the variable used must be defined as a symbolic variable.

Input the following script or open the script “**developper_et_factoriser.m**”:

```

expanding_factoring_expression_US.m    +
1 %definition af x as a symbolic variable
2 syms x
3 %% expandind function
4 expr1=(x-8)*(x+2)^2
5 expand(expr1)
6
7 %% factoring function
8 expr2=x^3 - 4*x^2 - 28*x - 32
9 factor(expr2,x)
10

```

Ln 1 Col 1

Figure 233: expanding and factoring an expression

Executing the script provides the following result:

```
>> expanding_factoring_US

expr1 =
(x + 2)^2*(x - 8)

ans =

x^3 - 4*x^2 - 28*x - 32          (expansion of expr1)

expr2 =

x^3 - 4*x^2 - 28*x - 32

ans =

[x - 8, x + 2, x + 2]           (factorization of expr2)
```

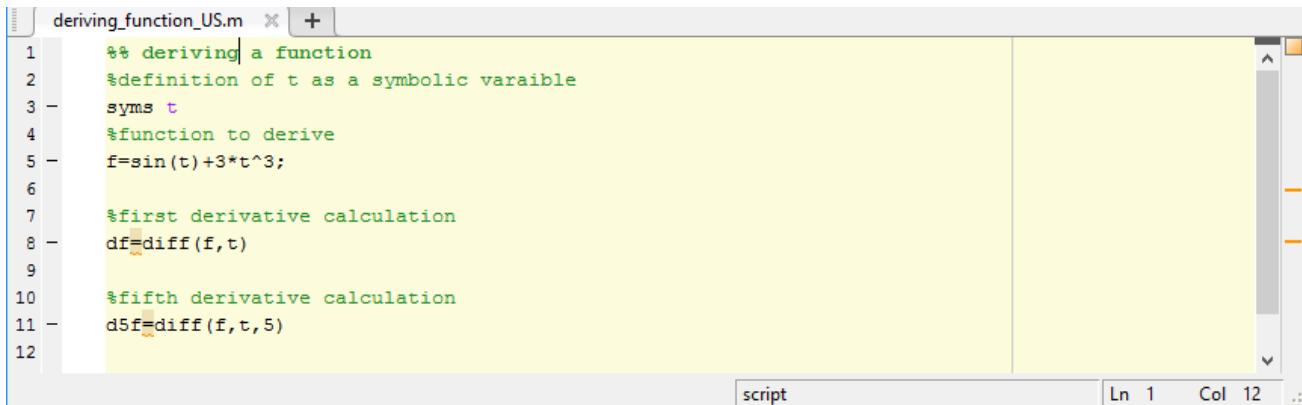
These commands can be useful for manipulating transfer functions.

3. Deriving a function

Consider the function: $f(t) = \sin(t) + 3t^3$.

We want to calculate the successive derivatives of this function in relation to the variable t .

Input the following script or open the script “**deriving_function_US.m**”:



```
deriving_function_US.m
1 %& deriving| a function
2 %definition of t as a symbolic variable
3 - syms t
4 %function to derive
5 - f=sin(t)+3*t^3;
6
7 %first derivative calculation
8 - df=diff(f,t)
9
10 %fifth derivative calculation
11 - d5f=diff(f,t,5)
12
```

Figure 234: script for deriving a function

Executing the script provides the following result:

```
>> >> deriving_function_US

df =
cos(t) + 9*t^2          (calculation of the first derivative of the function)

d5f =
cos(t)                  (calculation of the fifth derivative of the function)
```

4. Integrating a function

Consider the function: $f(t) = \sin(t) + 3t^3$.

$$I = \int_{-\frac{\pi}{3}}^{\pi} f(t) dt$$

We want to calculate the primitive of $f(t)$ and the following integral:

Input the following script or open the script “**integrating_function_US.m**”:

The screenshot shows a MATLAB script editor window titled "integrating_function_US.m". The code in the editor is as follows:

```
1 %% Integrating a function
2 %definition of t as a symbolic variable
3 - syms t
4 %% Integrating
5
6 %primitive calculation
7 - int(f,t)
8
9 %define integrale calculation
10 - int(f,t,-pi/3,pi)
```

The status bar at the bottom right indicates "script", "Ln 4", and "Col 13".

Figure 235: script used to integrate a function and calculate a defined integral

Executing the script provides the following result:

```
>> integrating_function_US
ans =
(3*t^4)/4 - cos(t)                                calculation of the primitive of f(t)
ans =
I = ∫ f(t) dt
(20*pi^4)/27 + 3/2                                calculation of
```

5. Using Laplace transforms

The *laplace* command enables the user to easily obtain the Laplace transform of a function.

Input the following script or open the script “**laplace_transform_US.m**”:

The screenshot shows a MATLAB script editor window titled "laplace_transform_US.m". The code in the editor is as follows:

```
1 %% Laplace Transform
2 - syms t;
3 - syms w;
4 - syms a;
5 - laplace(sin(w*t))
6 - laplace(cos(w*t))
7 - laplace(exp(-a*t)*cos(w*t))
8 - laplace(exp(-a*t)*sin(w*t))
```

Figure 236: script for calculating the Laplace transform of functions

Executing the script provides the following result:

```
>> laplace_transform_US

ans =
w/(s^2 + w^2)           calculation of the Laplace transform of sin(ωt)

ans =
s/(s^2 + w^2)           calculation of the Laplace transform of cos(ωt)

ans =
(a + s)/((a + s)^2 + w^2) calculation of the Laplace transform of e^-at cos(ωt)

ans =
w/((a + s)^2 + w^2)     calculation of the Laplace transform of e^-at sin(ωt)
```

6. Using inverse Laplace transform

The *ilaplace* command enables the user to easily obtain the inverse Laplace transform of a function.

Input the following script or open the script **transformee_inverse_laplace.m**:



```
inverse_laplace_transform_US.m
1 %& Symbolic variables
2 syms t;
3 syms w;
4 syms a;
5
6 %% Inverse Laplace Transform
7 syms s
8 ilaplace(s/(s^2 + w^2))
9 ilaplace(3/(2+s)-5/(3+s)^2)
10
11
```

Figure 237: script used to obtain the inverse Laplace transform of a function

Executing the script provides the following result:

```
>> inverse_laplace_transform_US

ans =
cos(t*w)           calculation of the inverse Laplace transform of  $\frac{s}{s^2 + \omega^2}$ 
ans =
3*exp(-2*t) - 5*t*exp(-3*t) calculation of the inverse Laplace transform of
      3
      5
      --- - ---
      2 + s    3 s^2
```

7. Partial fraction decomposition

The *residue* command enables the user to decompose a rational fraction into simpler elements.

Let's consider the function $H(s) = \frac{4s+5}{s^2 + 6s + 8}$ for partial fraction decomposition.

Input the following script or open the script **partial_fraction_decomposition_US.m**:

```
partial_fraction_decomposition_US.m
1 % Partial fraction decomposition
2 %fraction definition a/b
3 a=[4 5];
4 b=[1 6 8];
5
6 %decomposition
7 [r p k]=residue(a,b)
8
```

Figure 238: script for partial fraction decomposition

Executing the script provides the following result:

```
>> partial_fraction_decomposition_US
```

```
r =
```

```
5.5000
-1.5000
```

```
p =
```

```
-4
-2
```

```
k =
```

```
[]
```

The variable **r** contains the coefficients of the numerator for each simple element.

The variable **p** contains the values that cancel out the denominator for each of the simple elements.

The variable **k** contains the coefficients of the polynomial which will eventually be added to the simple elements to balance with the initial rational fraction.

The result for the example shown is thus as follows:

$$H(s) = \frac{4s+5}{s^2 + 6s + 8} = \frac{5.5}{p+4} - \frac{1.5}{p+2}$$

1. Solving a first order differential equation

It's very easy to solve linear differential equations.

Let us take the classic form of the first order linear differential equation as an example:

$$T \frac{df(t)}{dt} + f(t) = K$$

MATLAB can solve this equation using a symbolic calculation.

Definition of variables and the symbolic function:

```
>> syms K;syms T;syms t;syms f(t);
```

Definition of successive derivatives of $f(t)$

```
>> D1f=diff(f,1); D2f=diff(f,2);
```

Definition of the first order differential equation

```
>> equ1=T*D1f+f(t)==K
```

Solving the equation without specifying initial conditions:

```
>> dsolve(equ1)
ans =
K - C1*exp(-t/T)
```

The solution $f(t) = K - C_1 e^{-\frac{t}{T}}$ is given in accordance with a constant C_1 which MATLAB is unable to determine as the initial condition has not been defined.

It is possible to solve this equation by specifying the initial condition:

```
>> dsolve(equ1,f(0)==0)
ans =
K - K*exp(-t/T)
```

The solution is given taking into account the initial condition: $f(t) = K \left(1 - e^{-\frac{t}{T}} \right)$

You can open the script “**first_order_differential_equation_resolution_US.m**” which includes all the commands used to solve a first order differential equation.

1. Solving a second order differential equation

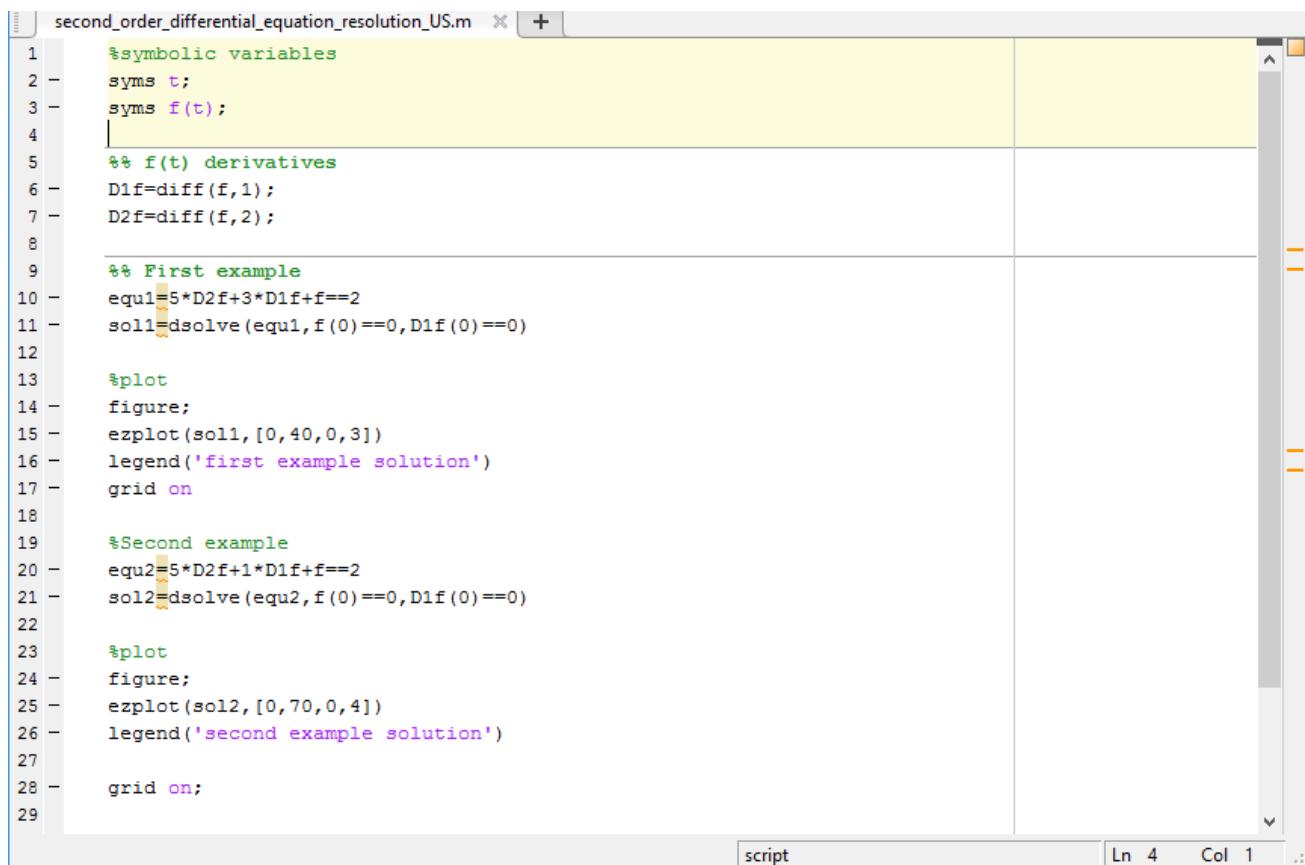
Now let us take the example of two second order linear differential equations:

$$5 \frac{d^2 f(t)}{dt^2} + 3 \frac{df(t)}{dt} + f(t) = 2 \quad \text{avec} \quad f(0)=0 \text{ et } \frac{df(0)}{dt}=0 \quad (\text{équation 1})$$

$$5 \frac{d^2 f(t)}{dt^2} + \frac{df(t)}{dt} + f(t) = 2 \quad \text{avec} \quad f(0)=0 \text{ et } \frac{df(0)}{dt}=0 \quad (\text{équation 2})$$

MATLAB can solve these equations and plot the function solutions.

Input the following script or open the script “**second_order_differential_equation_resolution_US.m**”:



```
second_order_differential_equation_resolution_US.m
1 %symbolic variables
2 syms t;
3 syms f(t);
4
5 %% f(t) derivatives
6 D1f=diff(f,1);
7 D2f=diff(f,2);
8
9 %% First example
10 equ1=5*D2f+3*D1f+f==2;
11 sol1=dsolve(equ1,f(0)==0,D1f(0)==0);
12
13 %plot
14 figure;
15 ezplot(sol1,[0,40,0,3])
16 legend('first example solution')
17 grid on
18
19 %Second example
20 equ2=5*D2f+1*D1f+f==2;
21 sol2=dsolve(equ2,f(0)==0,D1f(0)==0);
22
23 %plot
24 figure;
25 ezplot(sol2,[0,70,0,4])
26 legend('second example solution')
27
28 grid on;
```

Figure 239: script for solving a second order differential equation

Run the script.

```
>> second_order_differential_equation_resolution_US

equ1(t) =

f(t) + 3*diff(f(t), t) + 5*diff(f(t), t, t) == 2

sol1 =

2 - (6*11^(1/2)*exp(-(3*t)/10)*sin((11^(1/2)*t)/10))/11 - 2*exp(-(3*t)/10)*cos((11^(1/2)*t)/10)

equ2(t) =

f(t) + diff(f(t), t) + 5*diff(f(t), t, t) == 2

sol2 =

2 - (2*19^(1/2)*exp(-t/10)*sin((19^(1/2)*t)/10))/19 - 2*exp(-t/10)*cos((19^(1/2)*t)/10)
```

The solutions `sol1` and `sol2` are expressed in a symbolic format and are plotted using the *ezplot* command (Figure 240 and Figure 241)

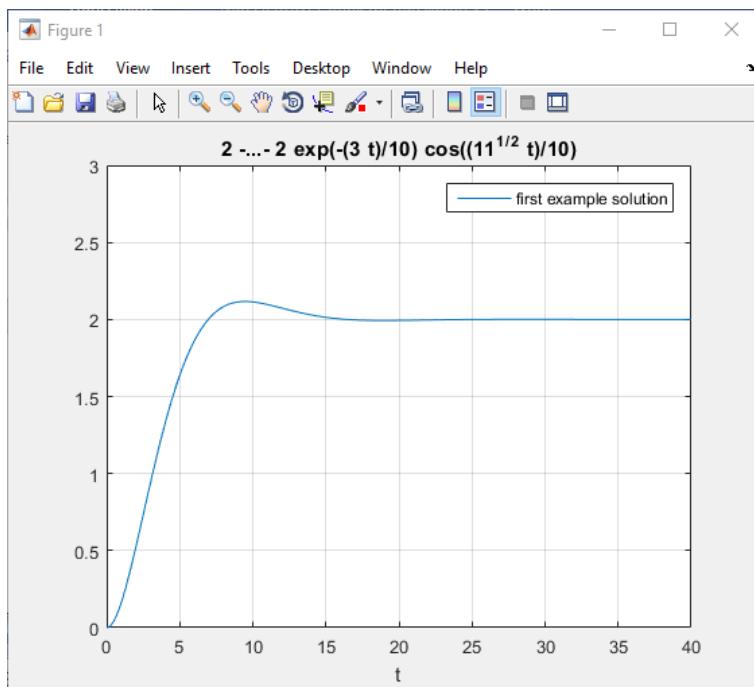


Figure 240: solution 1 for the second order differential equation

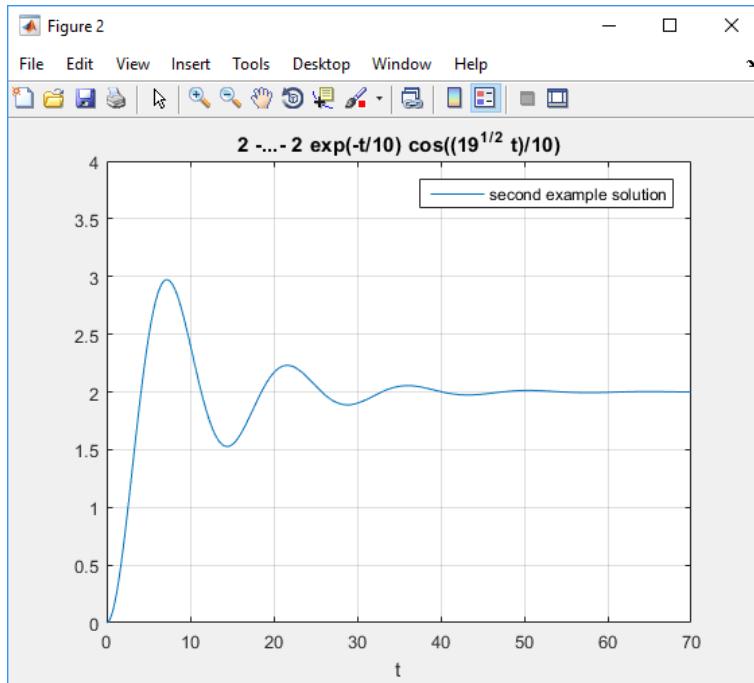


Figure 241: solution 2 for the second order differential equation

C. Manipulating transfer functions

MATLAB has many commands that enable the user to manipulate transfer functions and plot the time and frequency responses of systems.

1. Creating a transfer function

Consider two transfer functions:

$$\begin{cases} H(p) = \frac{4s+1}{2s^2+3s+6} \\ G(p) = \frac{100(s+1)(s+2)}{(s+0.1)(s+4)(s+10)} \end{cases}$$

There are several possible methods that can be used to create a transfer function, which vary in accordance with the form in which it is expressed.

For $H(p)$, the expression is expanded, and the most simple method is to directly input the numerator and denominator coefficients using the `tf` command

```
>> H=tf([4 1],[2 3 6])
```

H =

4 s + 1

2 s^2 + 3 s + 6

Continuous-time transfer function

For $G(p)$, the expression is factorized, and the simplest method is to directly input in this format to avoid having to expand it.

```
>> s=tf('s')
```

shows MATLAB that the variable s will be treated as a Laplace variable

s =

s

Continuous-time transfer function

```
>> G=100*((s+1)*(s+2))/((s+0.1)*(s+4)*(s+10))
```

G =

100 s^2 + 300 s + 200

s^3 + 14.1 s^2 + 41.4 s + 4

Continuous-time transfer function

The `zpk` command enables a transfer function to be input by indicating the poles and zeros.

It should be very simple to define $G(p)$ using this command

```
>> G=zpk([-1,-2],[-0.1,-4,-10],100)
```

G =

100 (s+1) (s+2)

(s+0.1) (s+4) (s+10)

Continuous-time zero/pole/gain model

2. Transfer function operations

We use the *pole* command to find the poles of a transfer function.

```
>> pole(G)
ans =
-10,0000
-4,0000
-0,1000
```

We use the *zero* command to find the zeros of a transfer function.

```
>> zero(G)
ans =
-2
-1
```

To calculate the equivalent transfer function of two transfer functions in series, we use the *series* command.

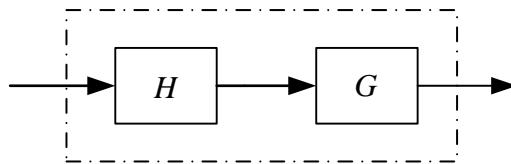


Figure 242: transfer functions in series

```
>> series(H,G)
ans =
200 (s+1) (s+2) (s+0.25)
-----
(s+0.1) (s+4) (s+10) (s^2 + 1.5s + 3)
Continuous-time zero/pole/gain model
```

This operation is equivalent to the command

```
>>H*G
```

To calculate the equivalent transfer function of two transfer functions in parallel, we use the *parallel* command.

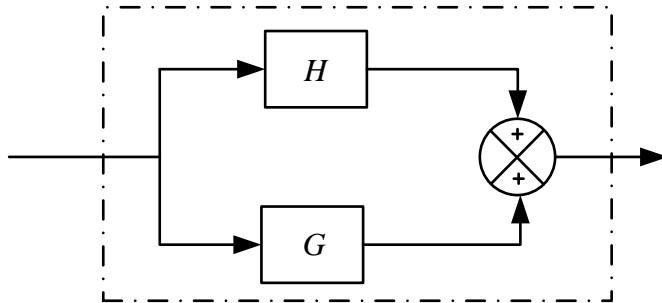


Figure 243: transfer functions in parallel

```
>> parallel(H,G)
ans =
102 (s+1.515) (s+1.334) (s^2 + 1.844s + 2.92)
-----
(s+4) (s+10) (s+0.1) (s^2 + 1.5s + 3)
Continuous-time zero/pole/gain model
```

To calculate a closed loop transfer function, we use the *feedback* command.

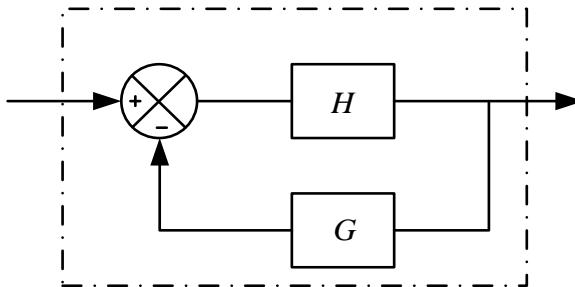


Figure 244: closed loop transfer function

```
>> feedback(H,G)
ans =
2 (s+0.25) (s+0.1) (s+4) (s+10)
-----
(s+0.2104) (s^2 + 3.028s + 2.392) (s^2 + 12.36s + 222.5)
Continuous-time zero/pole/gain model
```

We use the *inv* command to invert a transfer function.

```
>> inv(H)
ans =
2 s^2 + 3 s + 6
-----
4 s + 1
Continuous-time transfer function
```

Example of any transfer function

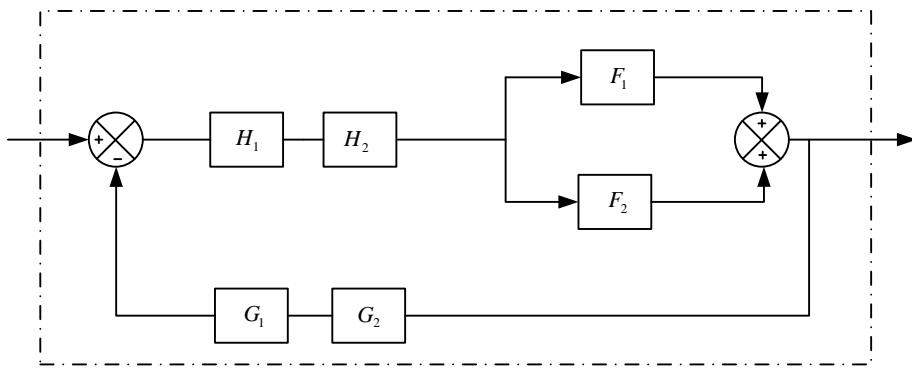


Figure 245: transfer functions of any system

It is possible to directly obtain the global transfer function for a system by inputting the command
`>>feedback((H1*H2)*parallel(F1,F2),G1*G2)`

The values of each of these functions should be entered first.

It is also very simple to construct the transfer function of a PID corrector by using the *pid* function.

```
>>pid(50,0.1,70)                               The three arguments representing the coefficients Kp, Ki and Kd of the corrector:  

pid(Kp,Ki,Kd)  

ans =  

    1  

Kp + Ki * --- + Kd * s  

      s  

with Kp = 50, Ki = 0.1, Kd = 70  

Continuous-time PID controller in parallel form.
```

3. Tracing the time responses of a system

We can use the *step* command to obtain the step response of a system

```
>>step(H); grid on
```

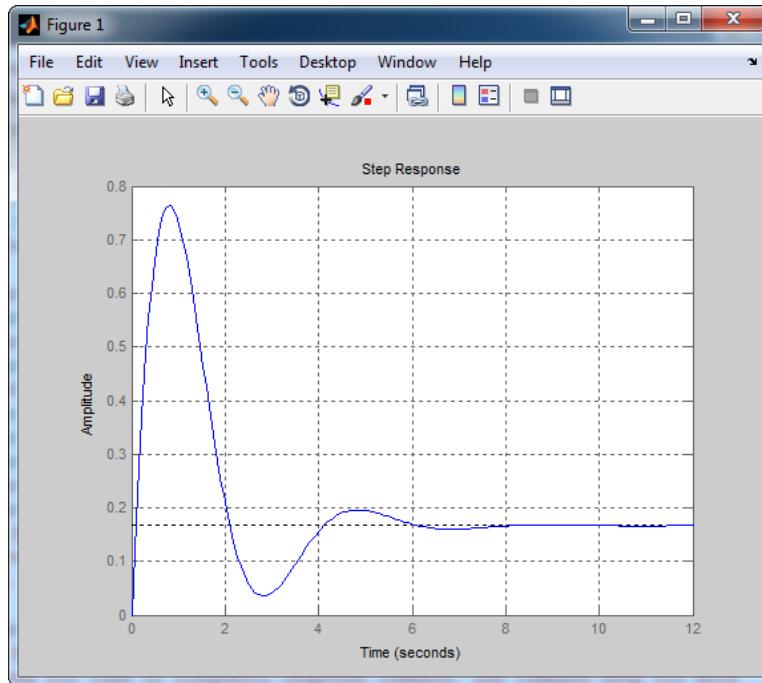


Figure 246: step response of a system

We can use the *impulse* command to obtain the impulse response of a system

```
>>impulse(H); grid on
```

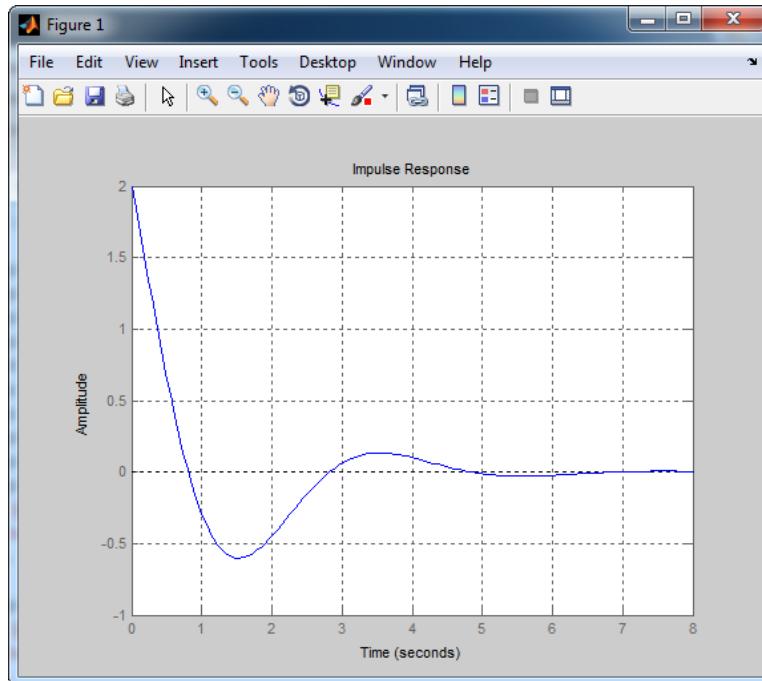


Figure 247: impulse response of a system

4. Tracing frequency responses of a system

We can use the *bode* command to obtain the Bode diagram for a system

```
>>bode(H); grid on
```

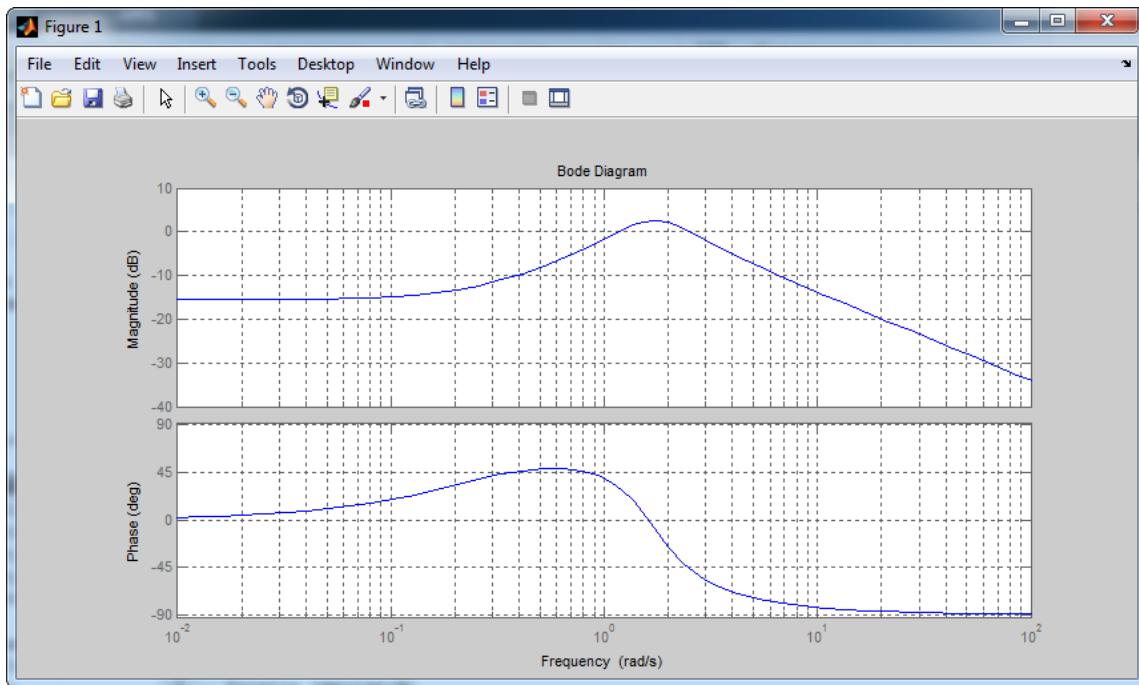


Figure 248: Bode diagram of a system

We can use the *nichols* command to obtain the Black-Nichols diagram for a system

```
>>nichols(H); grid on
```

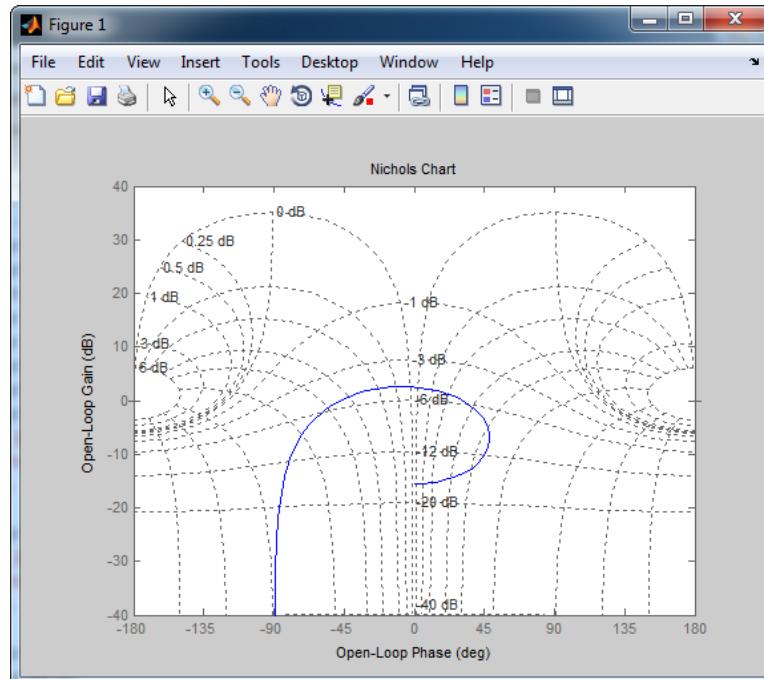


Figure 249: Black-Nichols diagram of a system

We can use the *nyquist* command to obtain the Nyquist diagram for a system

```
>>nyquist(H); grid on
```

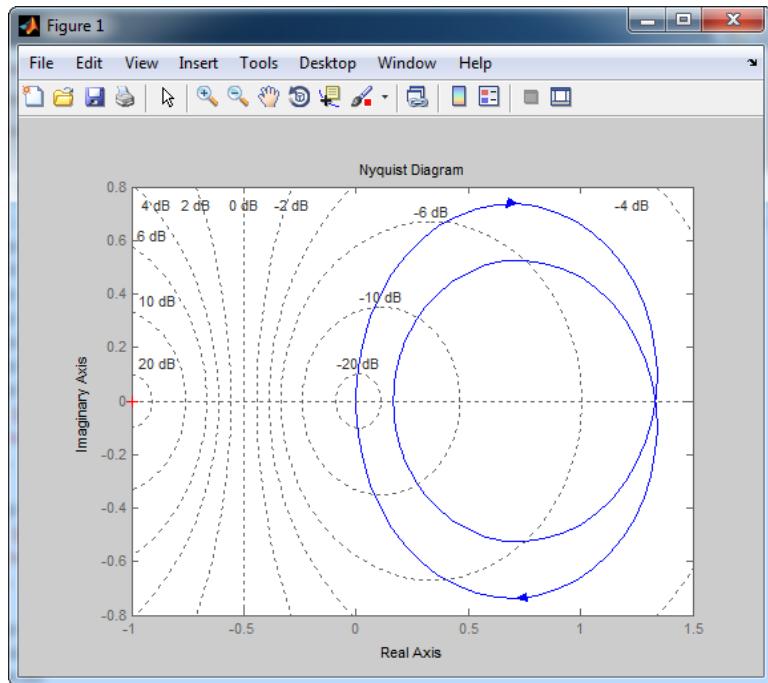


Figure 250: Nyquist diagram of a system

We can also plot two Bode diagrams on the same graph.

```
>>bode(H,G); grid on
```

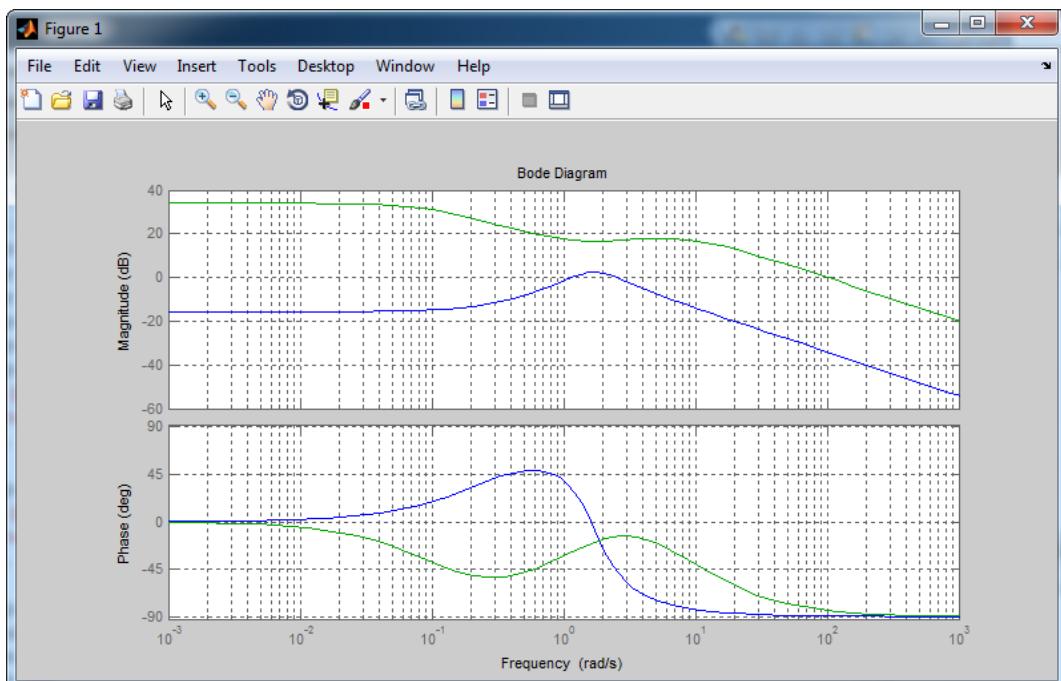


Figure 251: two Bode diagrams plotted on the same graph

5. Evaluate gain and phase margins

The gain and phase margins for an open loop transfer function can be obtained automatically using the `margin` command, which displays a Bode diagram that gives numerical and graphic information on the gain and phase margins.

```
>> margin(H)
```

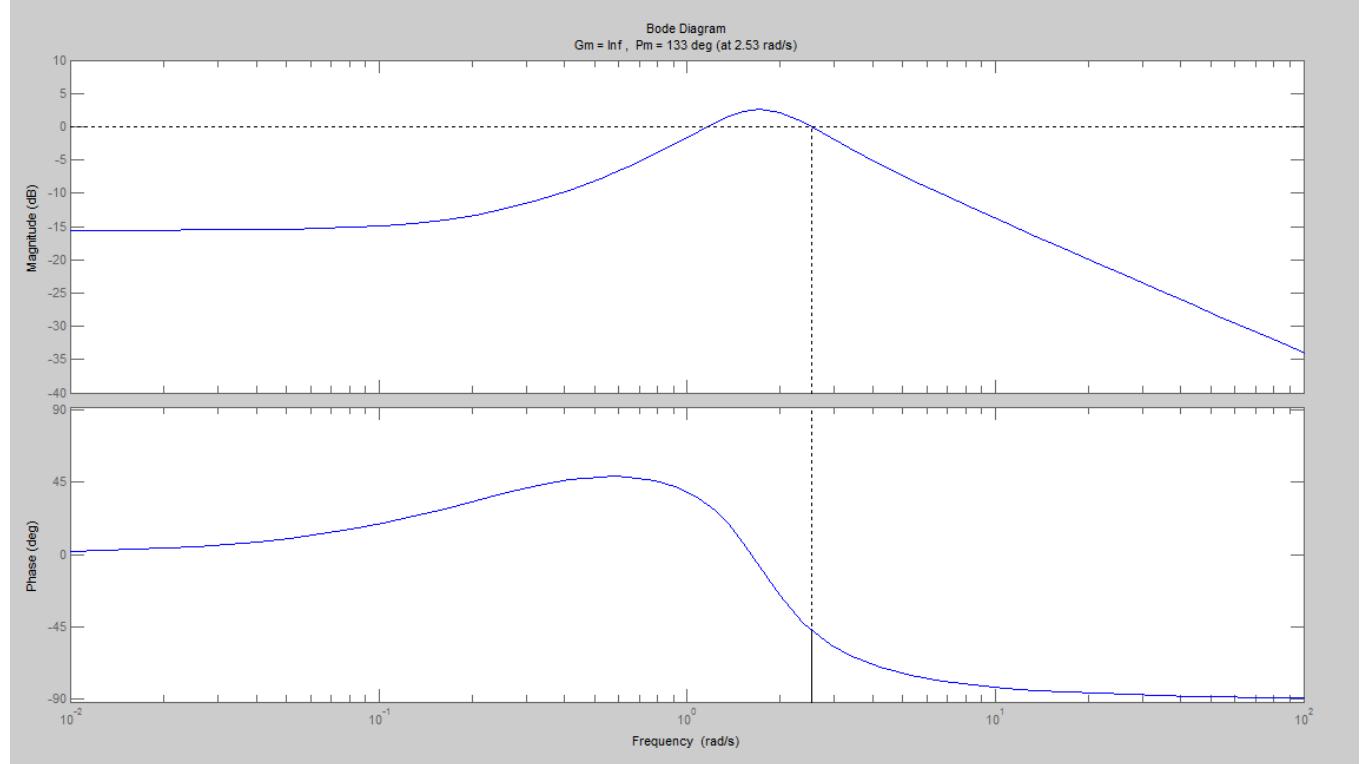


Figure 252: Bode diagram with indications of gain and phase margins

It is also possible to store the gain and phase margin values and the various pulses for which they are evaluated in the variables.

```
>> [gm,pm,wcg,wcp]=margin(H)
```

```
gm =
```

```
Inf
```

```
pm =
```

```
132.6226
```

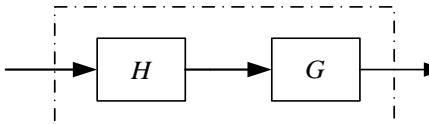
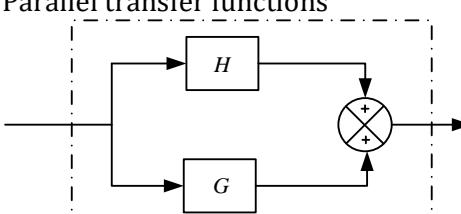
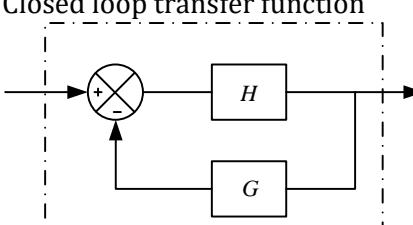
```
wcg =
```

```
NaN
```

```
wcp =
```

```
2.5255
```

6. Summary table of useful commands for transfer functions

Useful commands	
Functions	Commands
Creating a symbolic variable	<code>syms x</code>
Solving an algebraic equation	<code>solve(equation,x)</code>
Factoring an expression	<code>factor (x^3-4*x^2-28*x-32)</code>
Expanding an expression	<code>expand ((x-8)*(x+2)^2)</code>
Deriving a function	<code>diff (f,t)</code>
Integrating a function	<code>int (f,t)</code>
Solving a differential equation	<code>dsolve(equation)</code>
Calculating a defined integral	<code>int(f,t-pi/3,pi)</code>
Calculating the Laplace transform of a function	<code>laplace(sin(w*t))</code>
Calculating the inverse Laplace transform of a function	<code>ilaplace(3/(2+s)-5/(3*s)^2)</code>
Fraction decomposition of a rational fraction	<code>[r p k]=residue (a,b)</code>
Creating a transfer function based on the definition of polynomial coefficients	<code>H=tf([4 1],[2 3 6])</code>
Creating a transfer function based on knowledge of the gain, poles, and zeros	<code>G=zpk([-1,-2],[-0.1,-4,-10],100)</code>
Indicate that s is the Laplace variable	<code>s=tf('s')</code>
Directly create a transfer function after using the "s=tf('s')" function	<code>G=100*((s+1)*(s+2))/((s+0.1)*(s+4)*(s+10))</code>
Creating a transfer function for a PID corrector	<code>pid(Kp,Ki,Kd)</code>
Series transfer functions	<code>series(H,G)</code>
	
Parallel transfer functions	<code>parallel(H,G)</code>
	
Closed loop transfer function	<code>feedback(H,G)</code>
	
Invert a transfer function	<code>inv(H)</code>
Find the poles of a transfer function	<code>pole(H)</code>
Find the zeros of a transfer function	<code>zero(H)</code>
Step response	<code>step(H)</code>
Impulse response	<code>impulse(H)</code>
Bode diagram	<code>bode(H)</code>
Black-Nichols diagram	<code>nichols(H)</code>

Nyquist diagram	<code>nyquist(H)</code>
Multiple-plot diagrams	<code>bode(H,G)</code> or <code>nichols(H,G)</code>
Bode diagram with gain and phase margins	<code>margin(H)</code>
Store the gain and phase margin values and their corresponding pulses in the variables	<code>[gm,pm,wcg,wcp]=margin(H)</code>

To obtain further information on a command, you can receive **MATLAB** help by using the *doc* command.

```
>>doc bode          opens a help window about the "bode" command
```

Chapter 5: Simulink management

I. Introduction

This chapter uses an example project to present modelling carried out using **Simulink** and show the possible communications between the **MATLAB** and **Simulink** environments.

II. Controlling the temperature of an oven

This study looks at the servo control for the temperature of an industrial oven.



A heating element heats the oven chamber. A sensor informs the control board of the internal oven temperature. This measurement is compared to the temperature setpoint to calculate the difference. A proportional integral controller sends a voltage control to the heating element.

A. Opening the model

Open the file **oven_0_US.slx** to display the block diagram Figure 253 and explore the various blocks in the model.

This **Simulink** model represents the servo control for the temperature of an oven and the elements modelled by the various blocks are indicated in Figure 253.

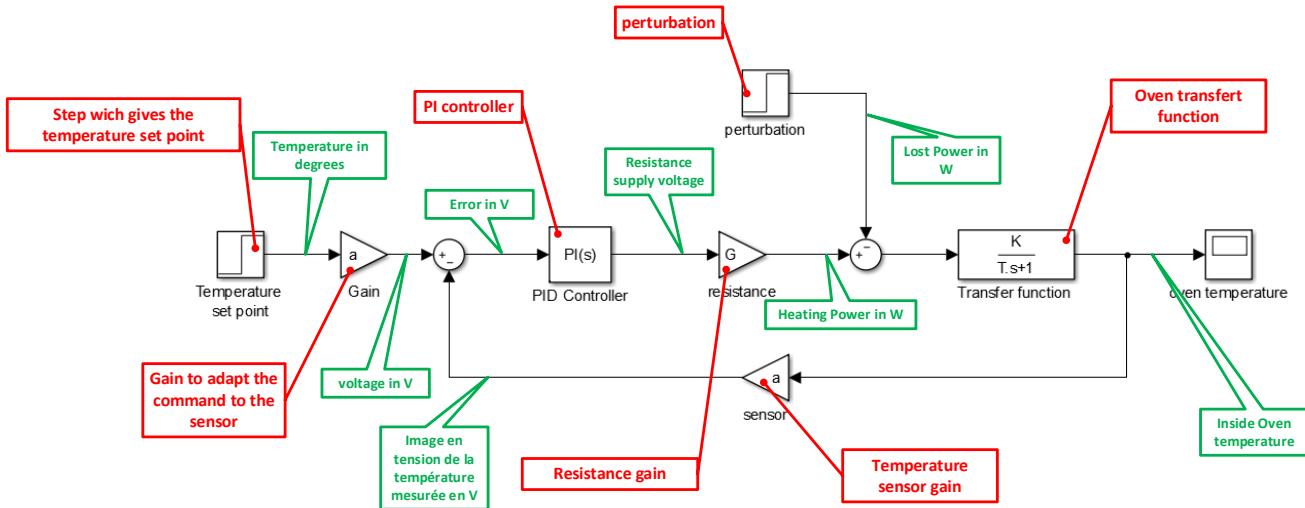


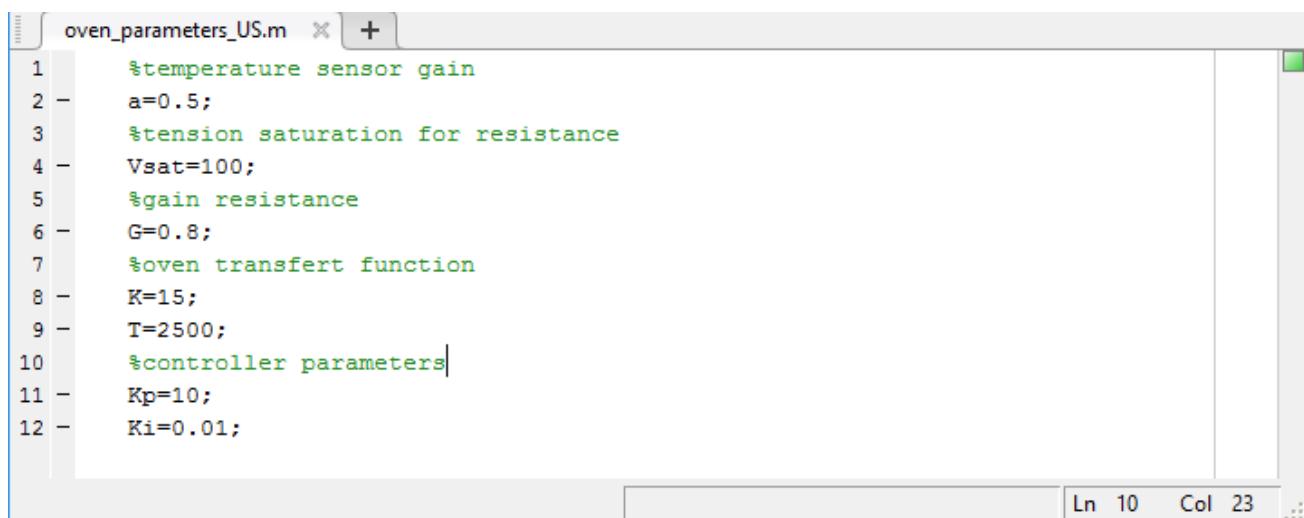
Figure 253: model of the servo control for the temperature of an oven

At this time, this model cannot be executed as all the physical dimensions of the model are represented by variables that must be defined in order to launch the simulation.

It can be helpful to group the values for all the physical dimensions of the system into a single script. Executing the script will enable all the variables to be created in the **MATLAB Workspace**. It will then be possible to launch the simulation in Simulink.

B. Opening the script containing the definition of the variables

Open the script **oven_parameters_US.m** by clicking on **Open script**



The screenshot shows a MATLAB code editor window with the script file "oven_parameters_US.m" open. The code defines various parameters:

```
1 %temperature sensor gain
2 a=0.5;
3 %tension saturation for resistance
4 Vsat=100;
5 %gain resistance
6 G=0.8;
7 %oven transfert function
8 K=15;
9 T=2500;
10 %controller parameters|
11 Kp=10;
12 Ki=0.01;
```

The code is color-coded: comments start with %, variable names are in blue, and numerical values are in black. The editor has a status bar at the bottom showing "Ln 10 Col 23".

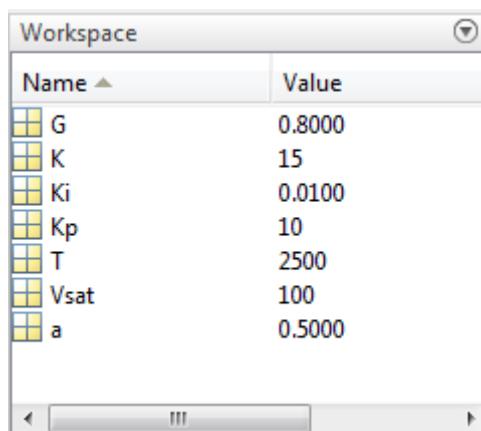
Figure 254: script containing simulation parameters

Run the script by clicking on **Run**



in the tool bar.

We can see that the variables were created and are visible in the **MATLAB Workspace**.



The screenshot shows the MATLAB Workspace browser window. It lists the variables defined in the script:

Name	Value
G	0.8000
K	15
Ki	0.0100
Kp	10
T	2500
Vsat	100
a	0.5000

Figure 255: visualizing the variables created in the Workspace

C. Launching the simulation

Launch the simulation of the Simulink model and visualize the temperature variation inside the oven

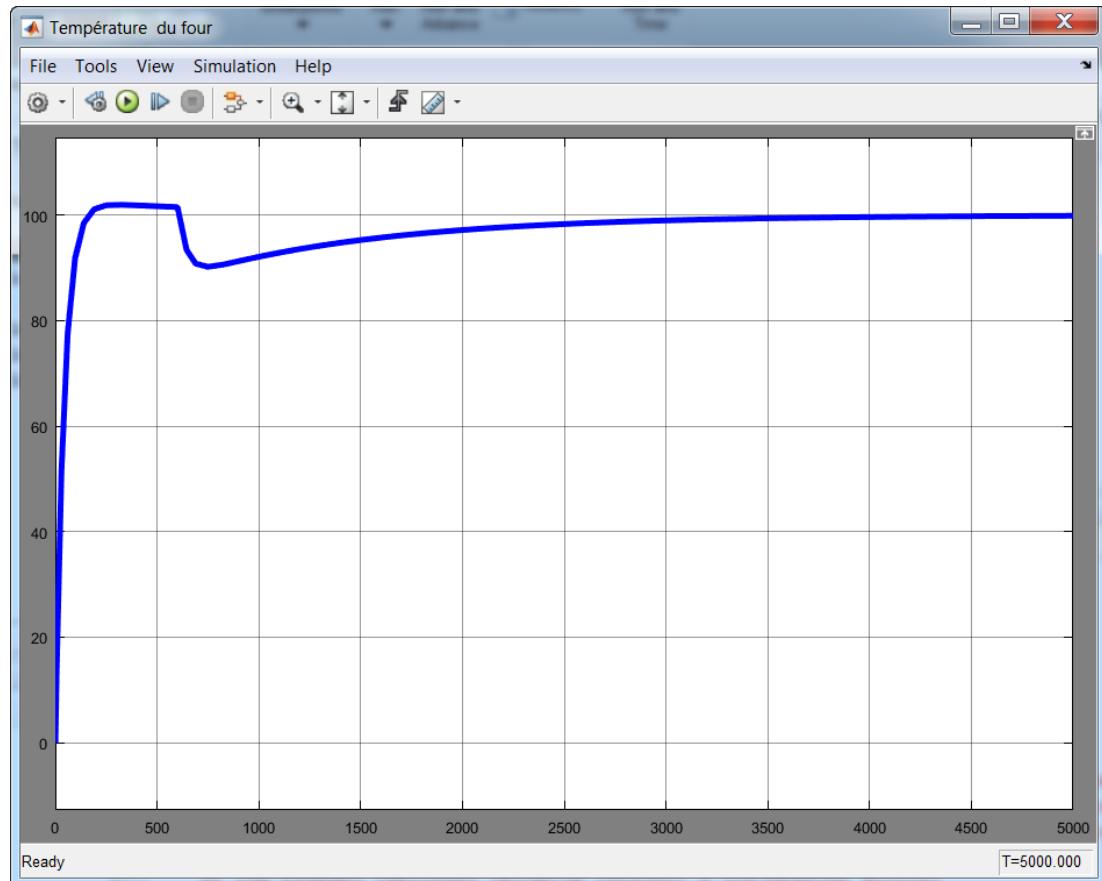


Figure 256: response of the oven temperature

A setpoint of 100° is set for the model. A disturbance is visible at $t=600\text{s}$ and is characterized by a fall in temperature. The proportional integral control enables the oven temperature to return to the setpoint level.

D. Plotting a Bode diagram with Simulink

There are many methods to plot a Bode diagram using a block diagram created in Simulink. The most simple method is to place **linearization points** on the block diagram and use the **Bode Plot** block from the **Simulink Control Design** library. There are different types of linearization points and we will see how to choose them to obtain both a closed loop and open loop Bode diagram.

Before beginning, it is best to give a name to the block diagram signals that will be used in the transfer functions to be plotted.

Name the signals following Figure 257:

- Input (Entrée)
- Output (Sortie)
- Difference (Ecart)
- Return (Retour)

To name a signal **right-click** on a signal and then choose **Properties**. The **Signal Properties** dialog box will open. The name of the signal can be entered into the **Signal Name** box.

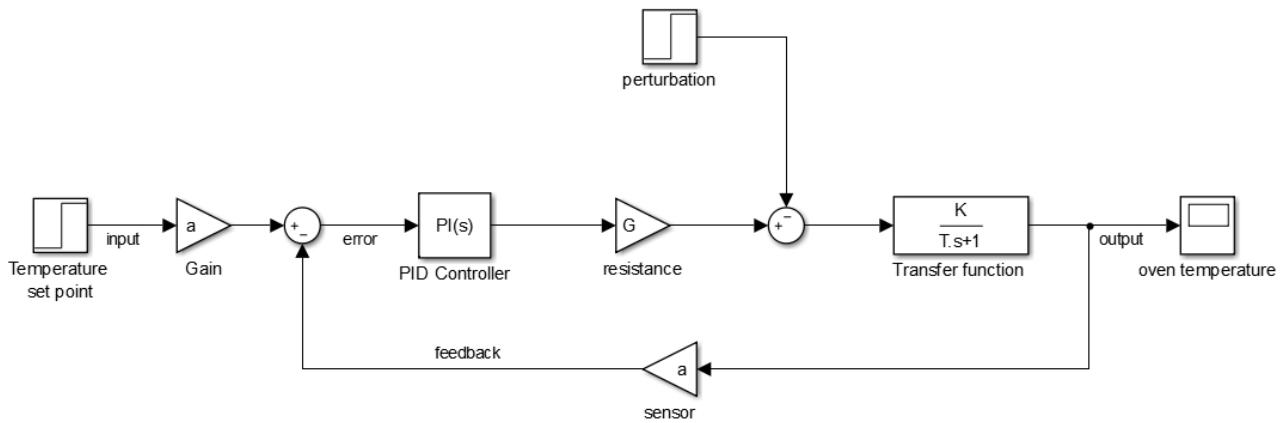


Figure 257: naming a Simulink signal

1. Plotting an open loop Bode diagram

To plot an open loop Bode diagram, two linearization points must be placed on the signals at the input and output of the open loop. In order to do this, two linearization points of the following type must be chosen:

- **Open loop input** (for the input of the open loop)
- **Open loop output** (for the output of the open loop)

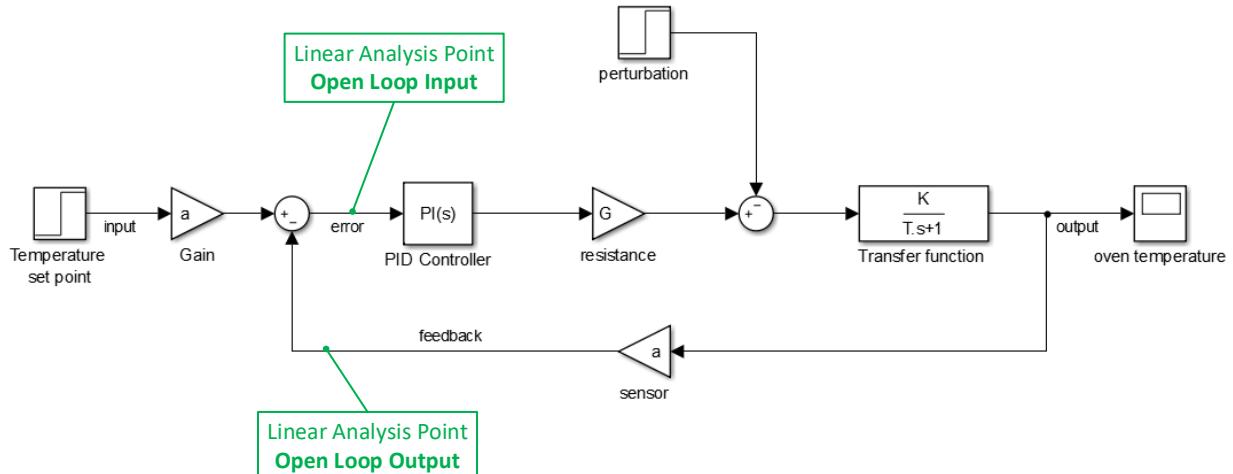


Figure 258: placing linearization points to plot an open loop Bode diagram

To place an **open loop input** on the “**error**” signal, **right-click** on the “**error**” signal, then choose **Linear Analysis point/open loop Input**.

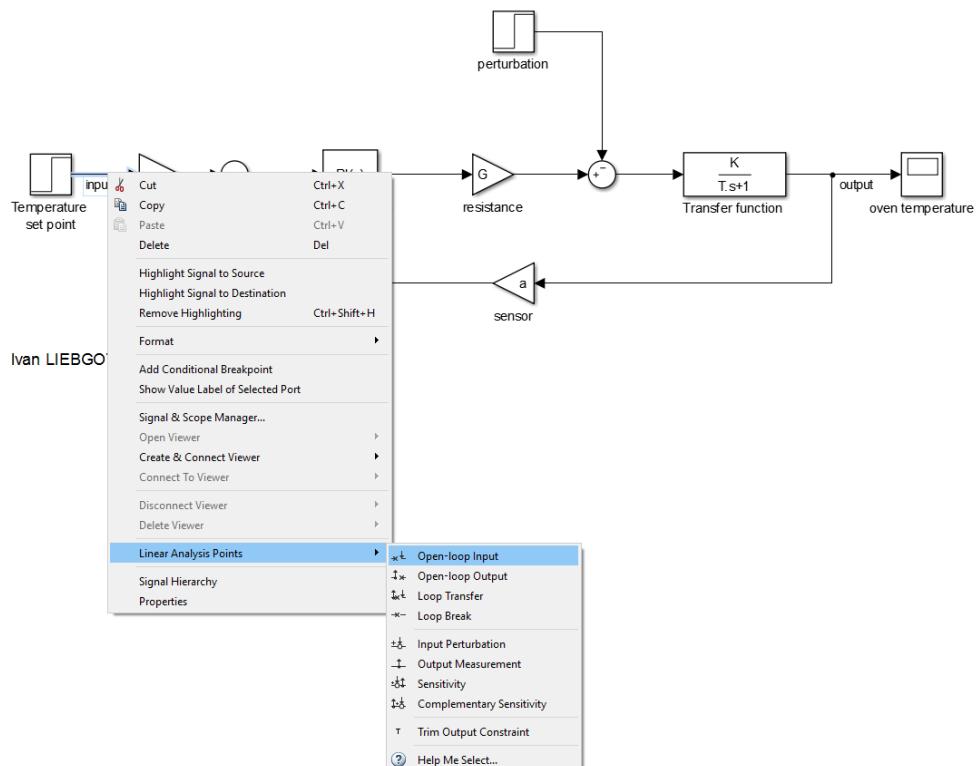


Figure 259: placing an “Open loop input” linearization point

To place an **open loop output** on the “retour” signal, right-click on the “retour” signal, then choose **Linear Analysis point/open loop Output**.

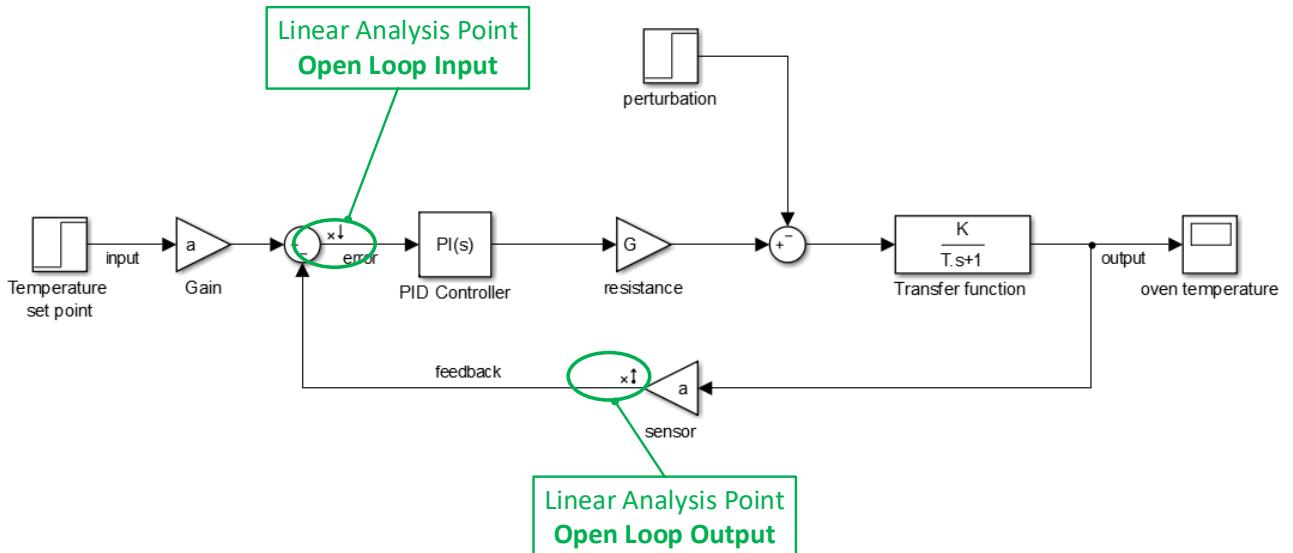


Figure 260: view of linearization points

Component function	View	Library
Plotting a Bode diagram	 Bode Plot	Simulink/Simulink Control Design/ Linear Analysis Plot

Insert a **Bode Plot** block into your model, and rename this block “**open loop**”.

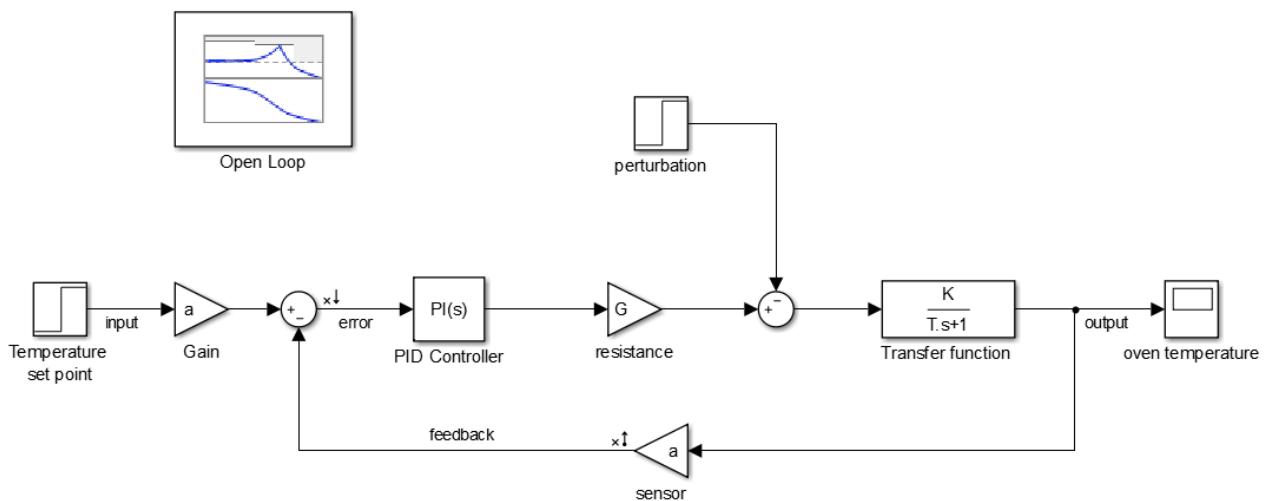


Figure 261: inserting a Bode Plot block

Double click on the block to open it.

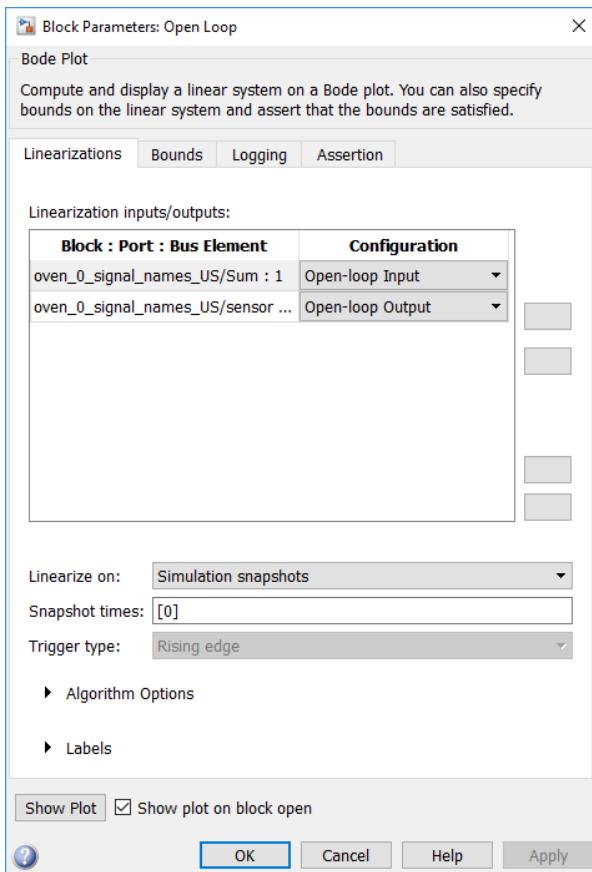


Figure 262: settings window for Bode Plot block

The two linearization points created will appear by default.
Click on **Show Plot** to display the plot window.

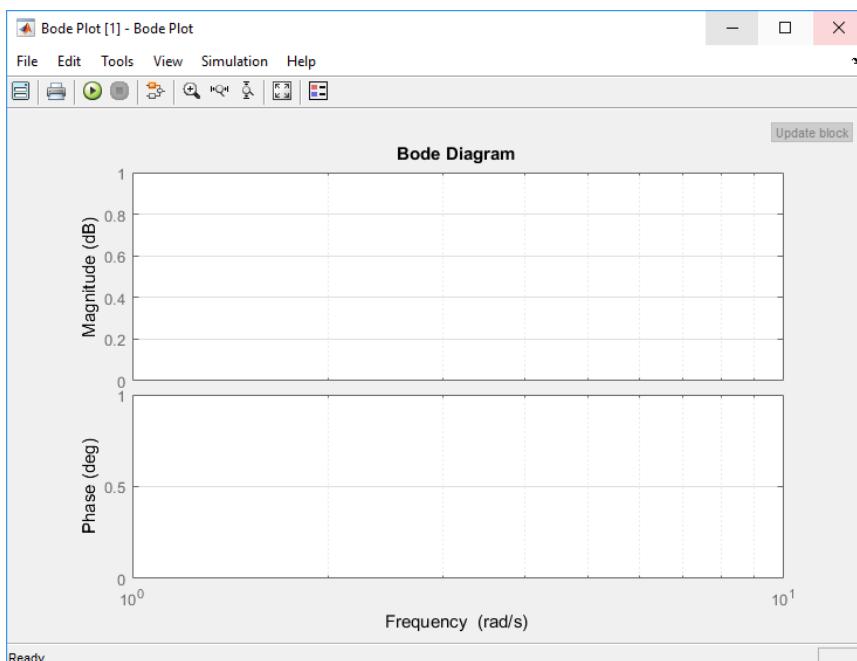


Figure 263: Bode diagram plot window



Click on Run to linearize the model. The Bode diagram will be plotted automatically.

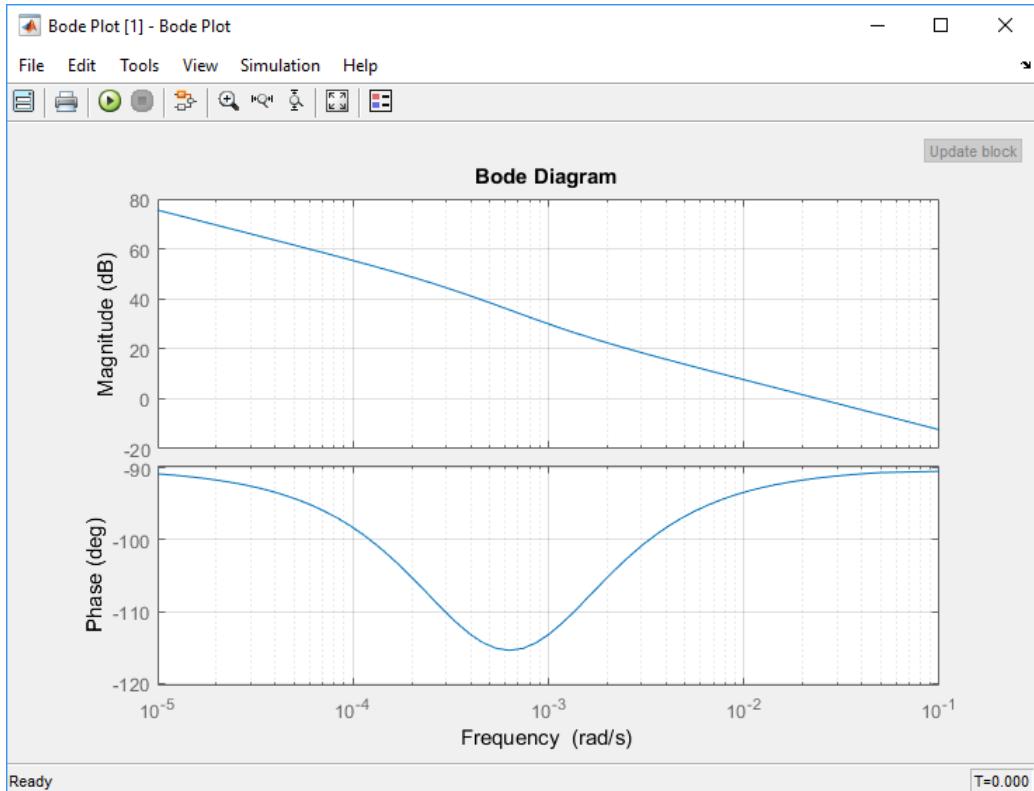


Figure 264: Bode diagram of the open loop

2. Plotting a closed loop Bode diagram

To plot a closed loop Bode diagram, two linearization points must be placed on the signals at the input and output of the closed loop. In order to do this, two linearization points of the following type must be chosen:

- **Input Perturbation** (for the input of the closed loop)
- **Output Measurement** (for the output of the closed loop)

To place an **Input Perturbation** type linearization point on the “input” signal, **right-click** on the “input” signal, then select **Linear Analysis point/Input Perturbation**.

To place an **Output Measurement** type linearization point on the “output” signal, **right-click** on the “output” signal, then select **Linear Analysis point/Output Measurement**.

The various linearization points appear on Figure 265.

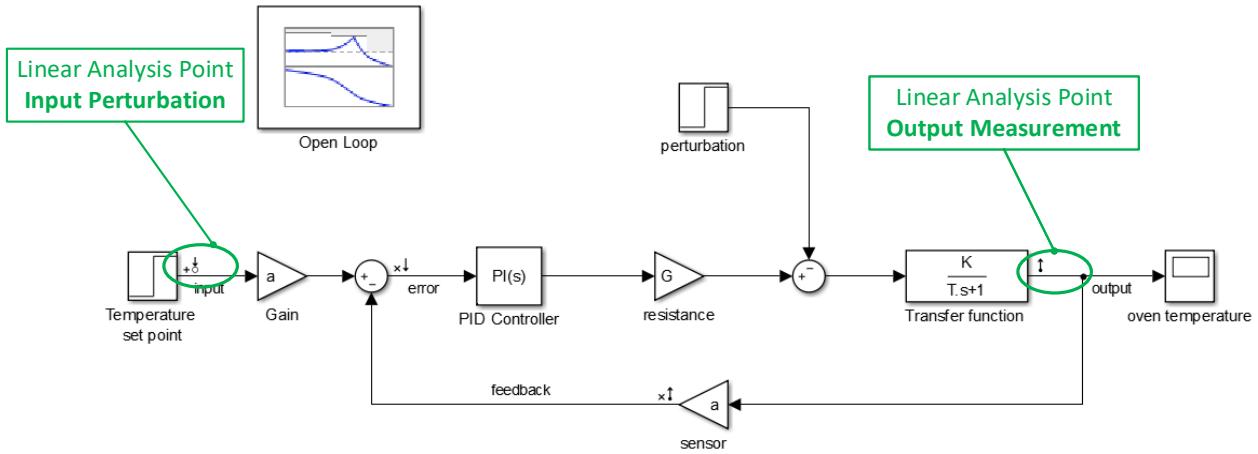


Figure 265: view of linearization points

Insert a second **Bode Plot** block into your model, and **rename** this block “**closed loop**”.

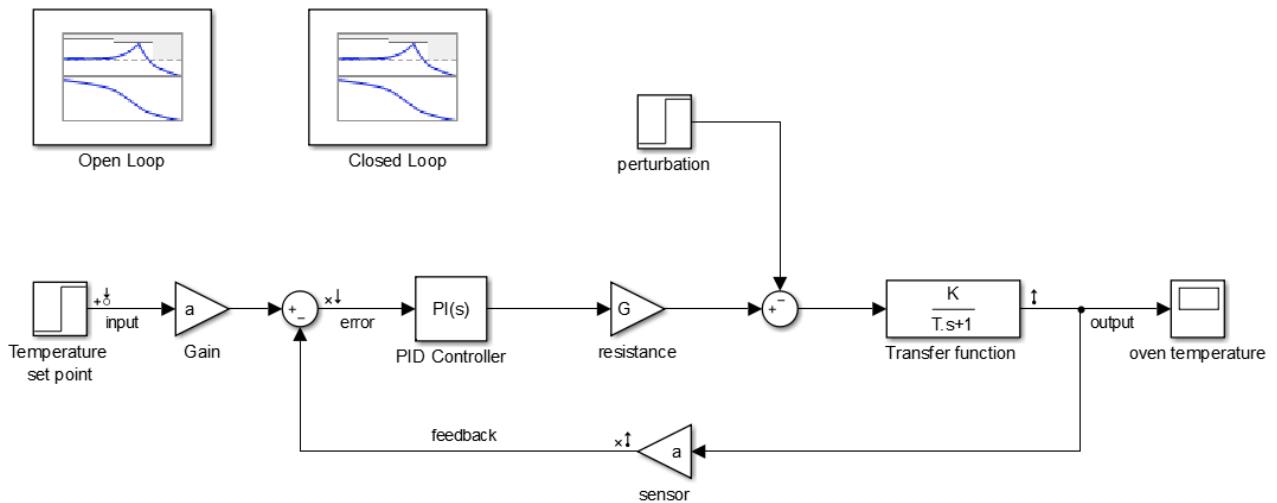


Figure 266: inserting a second Bode Plot block

Double click on the block to open it.

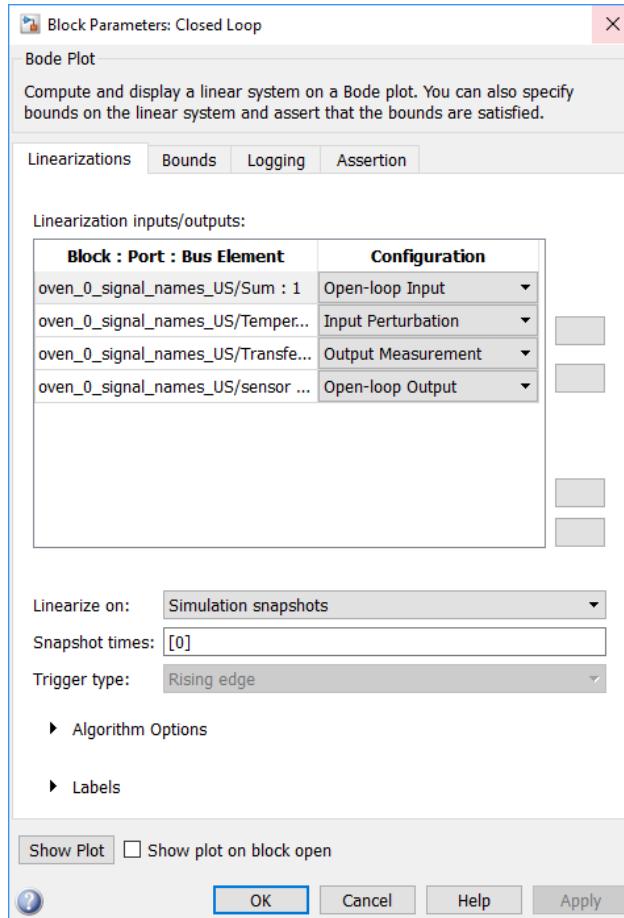


Figure 267: settings window for Bode Plot block

The two linearization points created will appear by default.

Keep the two linearization points that correspond to the closed loop and delete the two linearization points that correspond to the open loop.

To do this, select the linearization points to delete and use the **Delete Selected Linearization I/Os**



button to obtain the configuration in Figure 268.

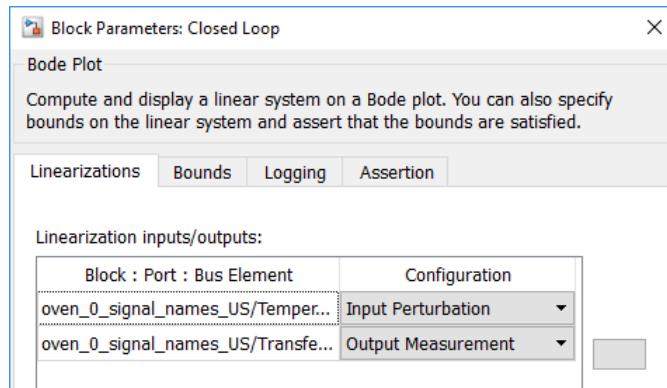


Figure 268: settings for the linearization points of the closed loop

Click on **Show Plot** to display the plot window.

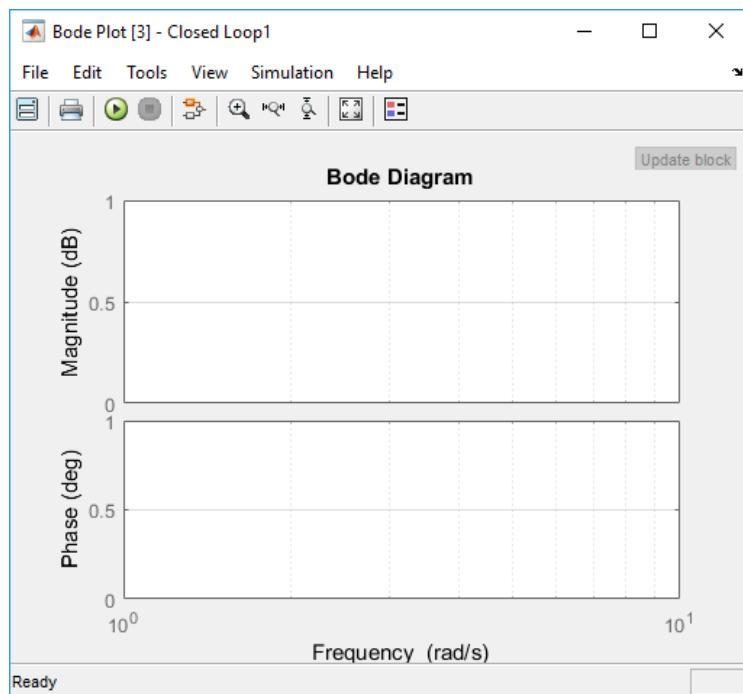


Figure 269: Bode diagram plot window

Click on **Run** to linearize the model. The Bode diagram will be plotted automatically.

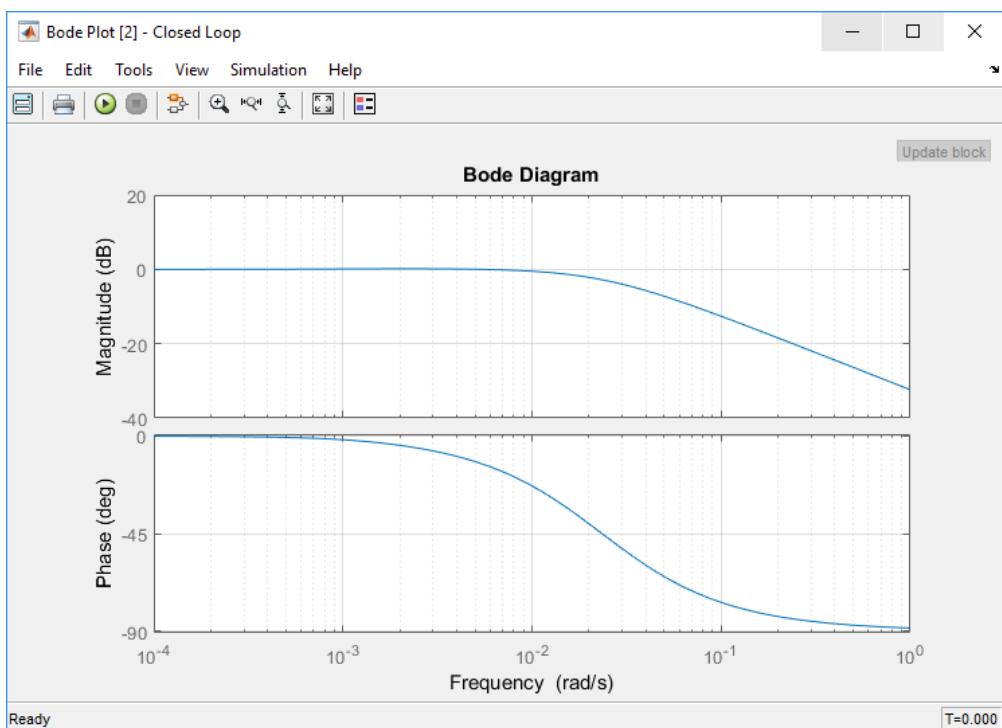
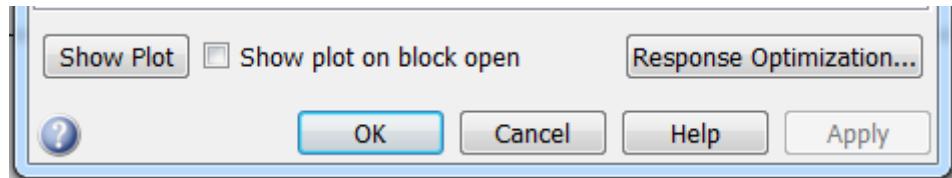


Figure 270: closed loop Bode diagram

If we want to open the graphic window for the Bode diagram by double clicking on the **Bode Plot** block, simply tick the **Show plot on block open** tick box.



We then have the Simulink model which we can modify and use to quickly visualise the rate of the Bode diagrams.

E. Plotting a Black-Nichols diagram

The method is almost the same as plotting a Bode diagram using the **Nichols Plot** block.

The file containing the complete model is available under the name **oven_0_Bode_Nichols_US.slx**

F. Adding and configuring a saturation

We are going to add a saturator to account for the 100V limit which represents the maximum voltage for controlling the heating element.

The non-linearities (threshold, hysteresis, etc.) of the Simulink library can be found in **Simulink/Discontinuities**.

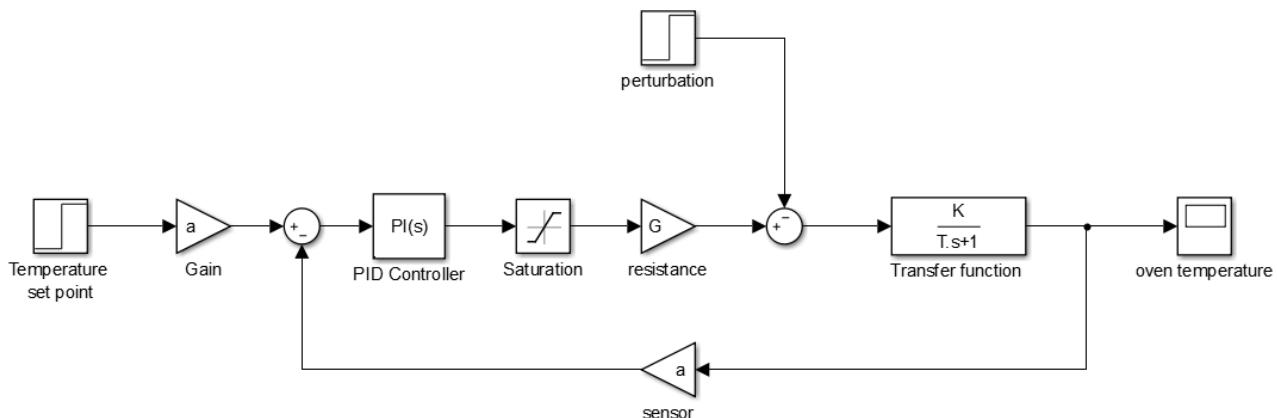


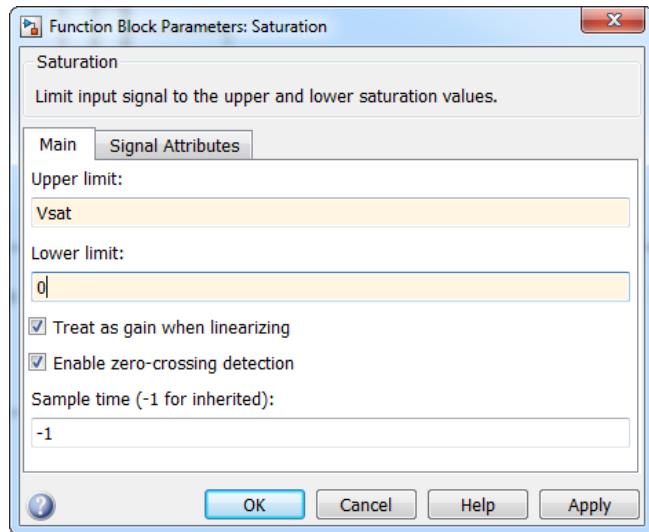
Figure 271: servo control for the temperature of an oven with saturation

Configuration



This component enables the output from the block to be limited between the two values for the minimum and maximum saturation.

Here the maximum value is given by the variable Vsat, with the minimum value left at 0.



Launch the simulation and view the response obtained with saturation.

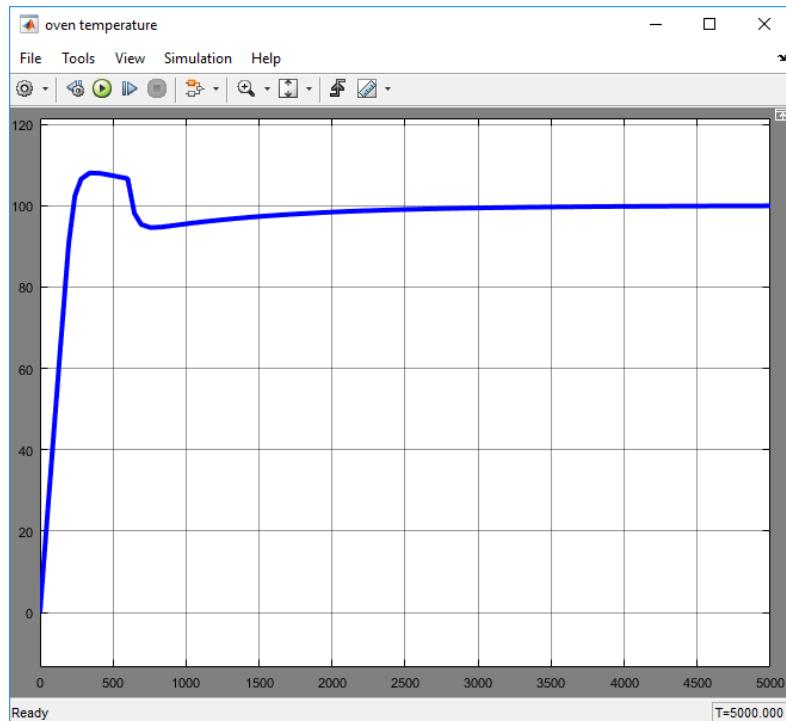
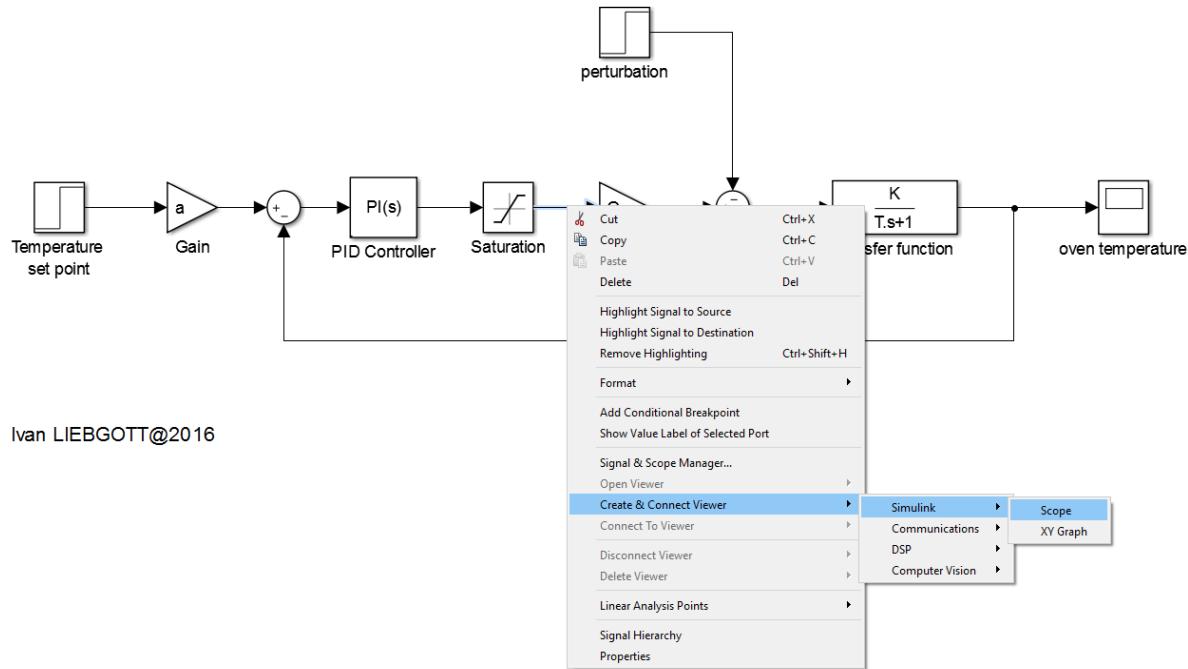


Figure 272: response of the oven temperature with saturation

We can see that the saturation slows the system, making the response time 5% higher. It is possible to place a classic scope in order to visualize the output voltage of the saturator. However, in order to visualize the signals without overloading the model, MATLAB offers another display mode that places a scope directly over a signal.

To do this, right click on the signal line, then select **Create&connect Viewer/Simulink/Scope**



A small scope will appear on the signal at the saturator output.

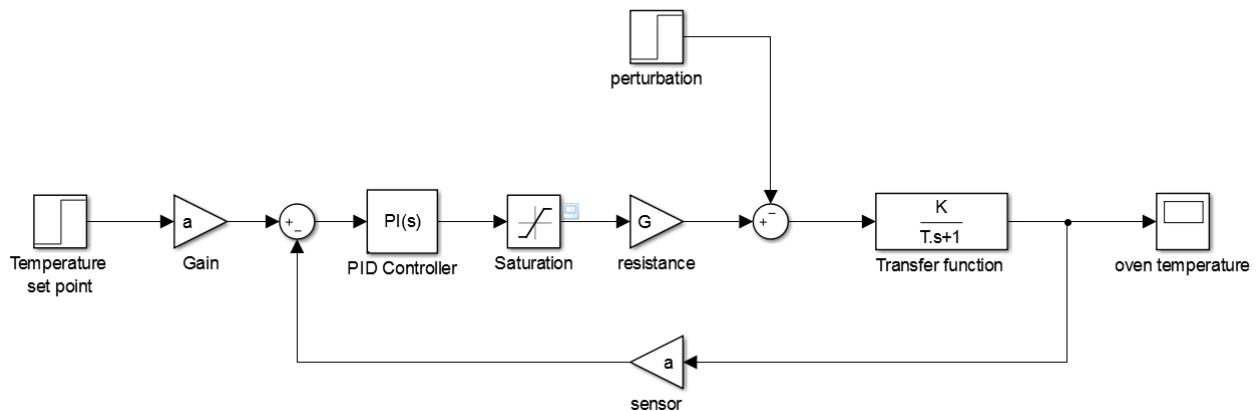


Figure 273: placing a scope over a signal

Relaunch the simulation and double click on the saturator scope to display the voltage output from the block.

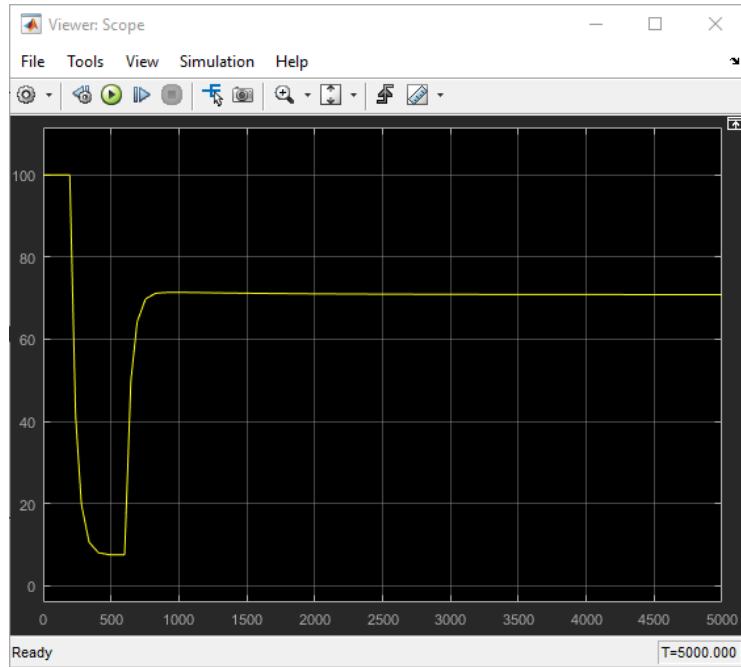


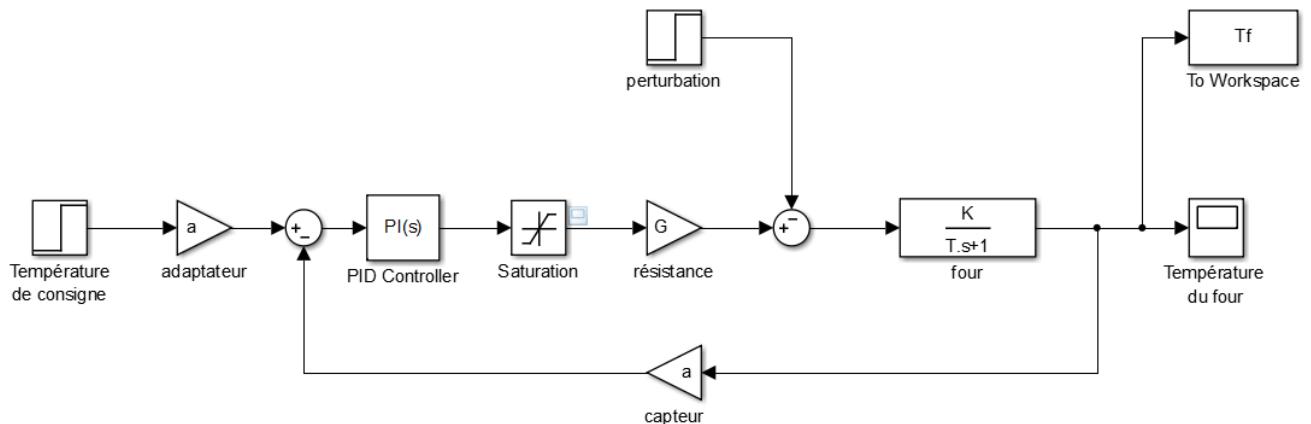
Figure 274: Displaying the output voltage from the saturator

We can see that voltage saturates at 100V for around 200s, which explains the slowdown in the system.

G. Exporting the simulation variables to the Workspace

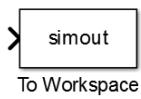
It can often be helpful to export variables containing simulation results into the **Workspace**. This enables all the functions of **MATLAB** to be used to present the results.

We want to plot a collection of curves that will enable us to see the influence of the **Kcor** gain of the proportional corrector on the oven temperature. In order to do this, we need to create a **Tf** variable that corresponds to the oven temperature and export this variable to Workspace using a **To Workspace** block that can be found in the **Simulink/Sinks** library.



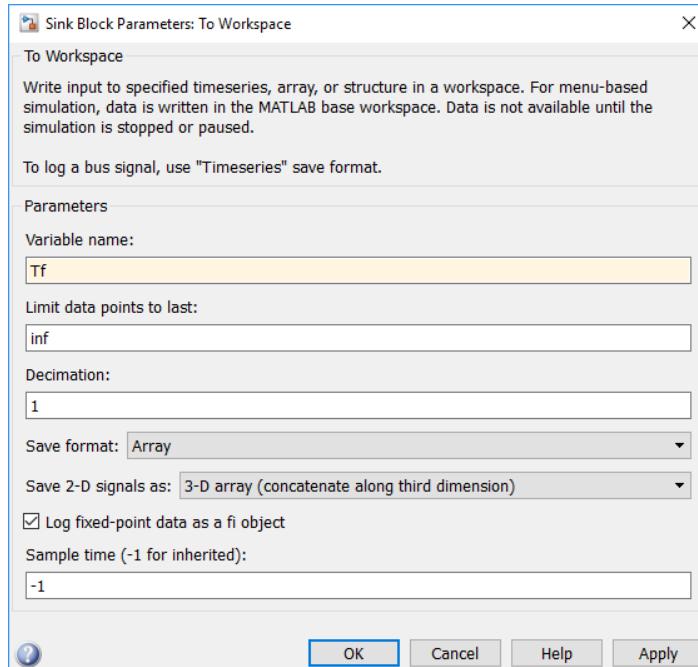
Configuration

Exporting variables to the Workspace



Simulink/Sinks

This component enables the user to create a variable for a **Simulink** signal and export it to the **MATLAB** Workspace. Simply specify the name of the variable (**Tf** in this instance) and its format. To export the variable in the form of a simple vector, choose the **Array** format in the "Save format" menu.



To then be able to plot the **Tf** variable over time, it requires a second vector representing the abscissa axes, which will be composed of different time values corresponding to the calculation steps of the solver. This variable is generated automatically by Simulink and exported to the Workspace. This variable is named **tout** (all) by default.

To modify the name of this **tout** (all) variable, **click** on the Simulink solver settings and choose the **Data Import/Export** tab on the left side of the window. Replace **tout** (all) with **t**.

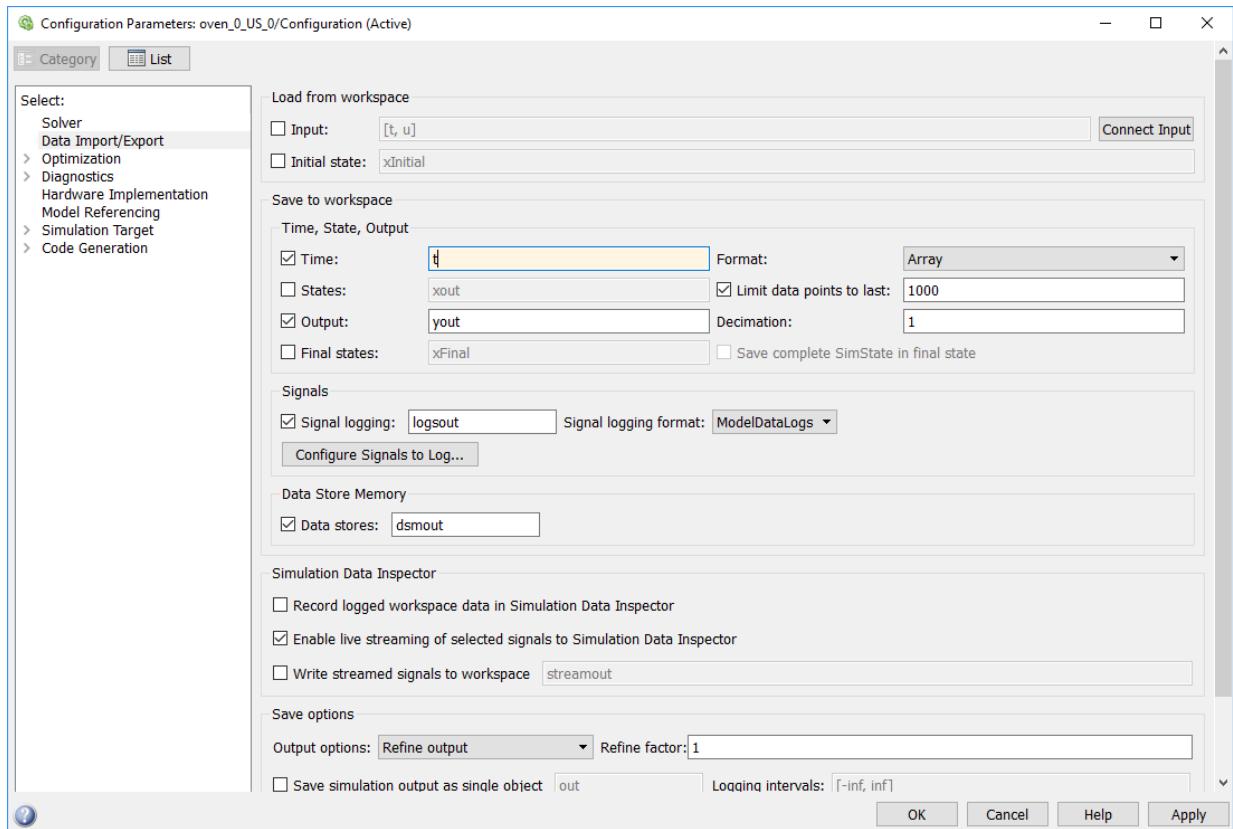


Figure 275: configuration window for exporting data to the Workspace

Save the model under the name ***oven_data_export_US.slx***.

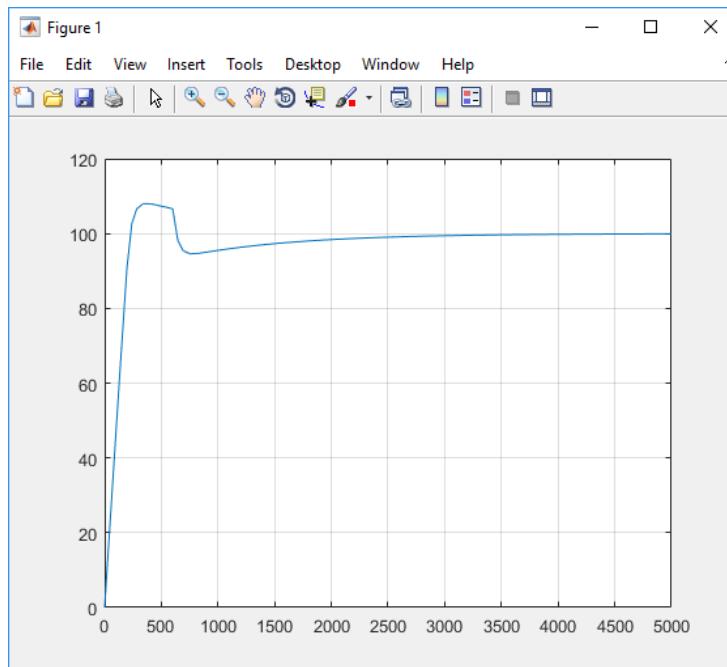
If necessary, the complete model is available in the file ***oven_data_export.slx***.

Launch the simulation.

Observe the **MATLAB** Workspace and check that the **t** and **Tf** variables have been created.

In the **MATLAB** command window type the following command:
`>>plot(t,Tf);grid on;`

A graph representing temperature over time will appear.



1. Writing a script to plot a series of curves

The **sim** command launches a simulation of a Simulink model from the command line. We are going to use this to write a script to display the influence of the gain of the proportional corrector.

Input the following script, then **save** and **run** it.

```
plot_oven_temperature_KcorUS.m
1 - hold all;
2 - grid on;
3 - % Kp is varying from 1 to 10
4 - % launch the simulation of the "oven_data_export_US" model
5 - for Kp=1:10
6 -     sim('oven_data_export_US');
7 -     plot(t,Tf,'LineWidth',2);
8 - end
9
```

Figure 276: script to plot a series of curves

The script is available in the file **plot_oven_temperature_Kcor_US.slx**.

You should obtain the following result:

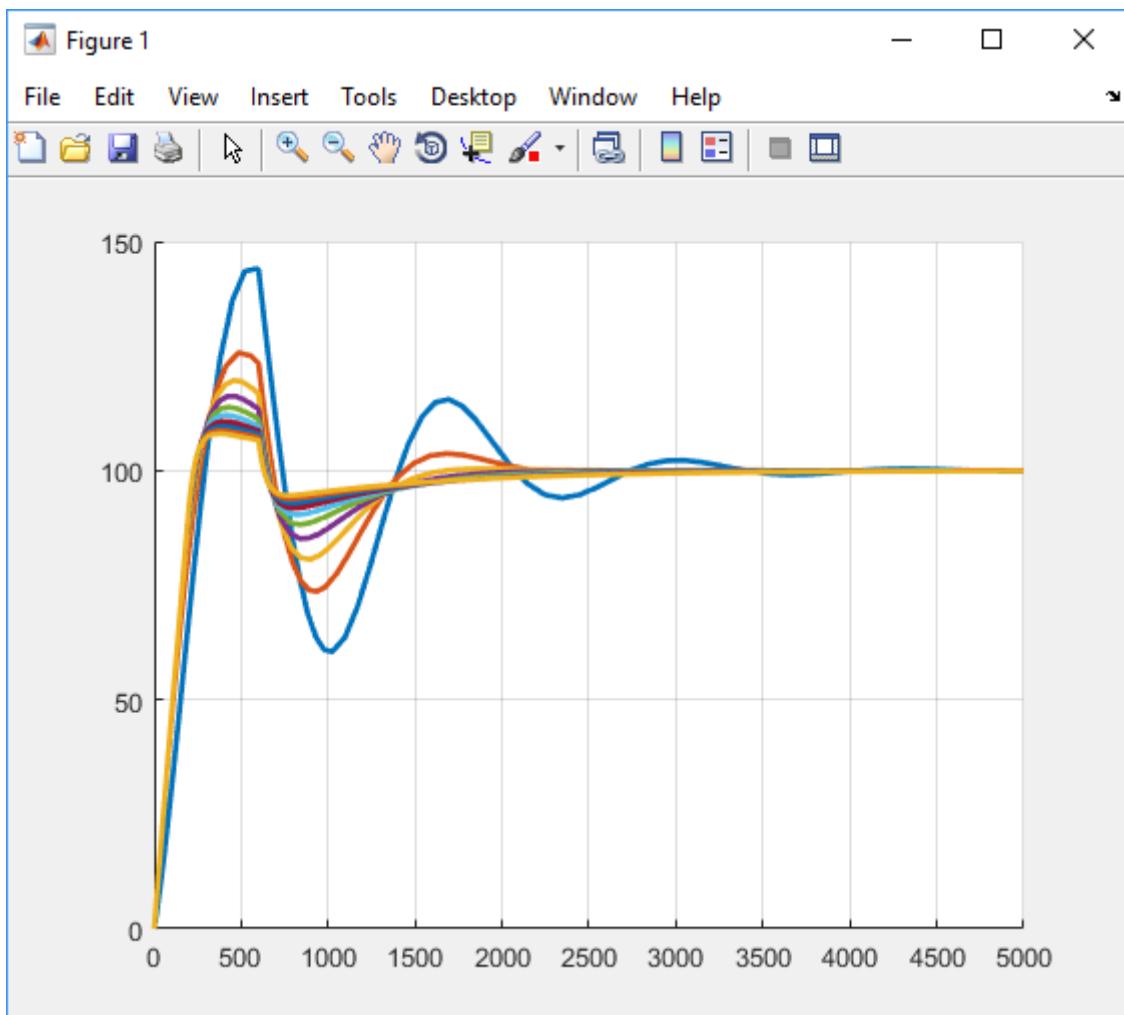


Figure 277: influence of proportional correction on the oven temperature

Chapter 6: Stateflow management

I. Introduction to Stateflow

Stateflow is a MATLAB module that models the sequential behavior of systems in the form of state machines.

A state machine is composed of “states” which represent the various modes of a system and “transitions” which translate passage conditions from one state to another. Upon activation of a state, actions can be performed by the system, which are then shown in the corresponding state.

A. Modelling a state machine reference course with Stateflow

We are going to create an elementary state diagram to model the logic of the course loop for the hydraulic pilot of a boat.

The system receives two input variables:

- **mes_cap**: the actual course of the boat measured using a compass
- **set_cap**: the reference course of the boat

The system returns an output variable:

- **set_barre**: angle setting of the boat helm

The state graph is composed of 3 distinct states:

“Pause” state

The system is in this state if there is a course error of less than 0.5° . The hydraulic pilot maintains the helm at 0° as long as the course error does not exceed 1° . If this threshold is exceeded, the system enters into the “Delay” state.

“Delay” state

The system is in this state if there is a course error exceeding the threshold of 1° . If the course error is sustained for 20s at greater than 1° , the system transitions into the “Cap_correction” state. If during these 20s the course error falls back below the threshold of 0.1° , the system returns to the “Pause” state.

“Cap_correction” state

The hydraulic pilot corrects the course of the boat and the helm setting is obtained based on the course error. The system changes from the “Cap_correction” state to the “Pause” state when the course error returns to a level below 0.05° .

B. Creating a state diagram

1. Opening the model

Open the model **hydraulic_pilot_stateflow_0_US.slx**.

This model contains the model of the hydraulic pilot for a boat, excluding the state diagram which controls the cap loop.

Double click on the “Simulink and Stateflow” sub-system and see the diagram to be completed as in Figure 278.

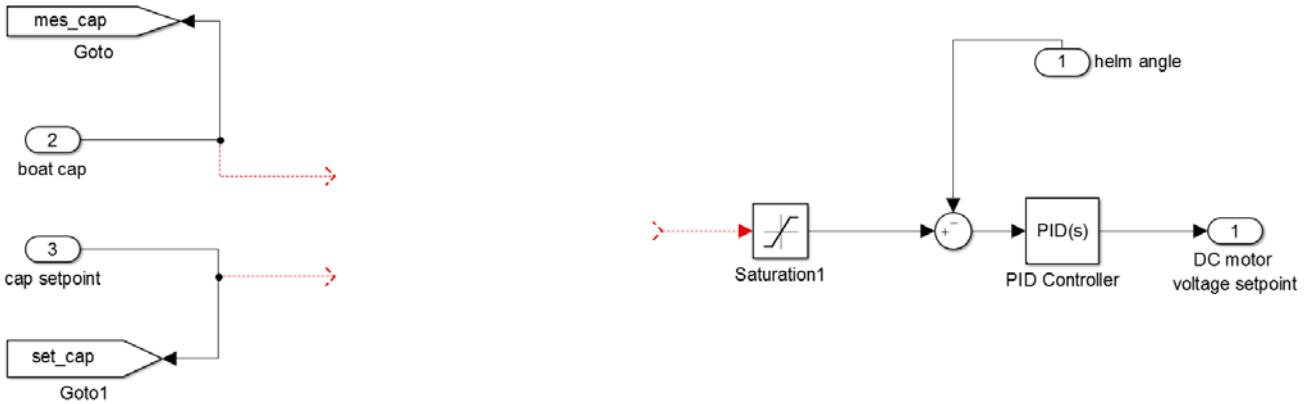


Figure 278: course loop without state diagram

On looking at the model, we can see that the cap loop does not include any command logic. We will model this command logic using a states diagram.

2. Inserting a “chart”

The component that enables the creation of a state diagram is called a “chart”. This “chart” will contain the states and transitions.

Insert a new “chart” from the **Stateflow** library.

Component function	View	Library
Chart		Stateflow

Position and resize the chart to obtain the configuration in Figure 279.

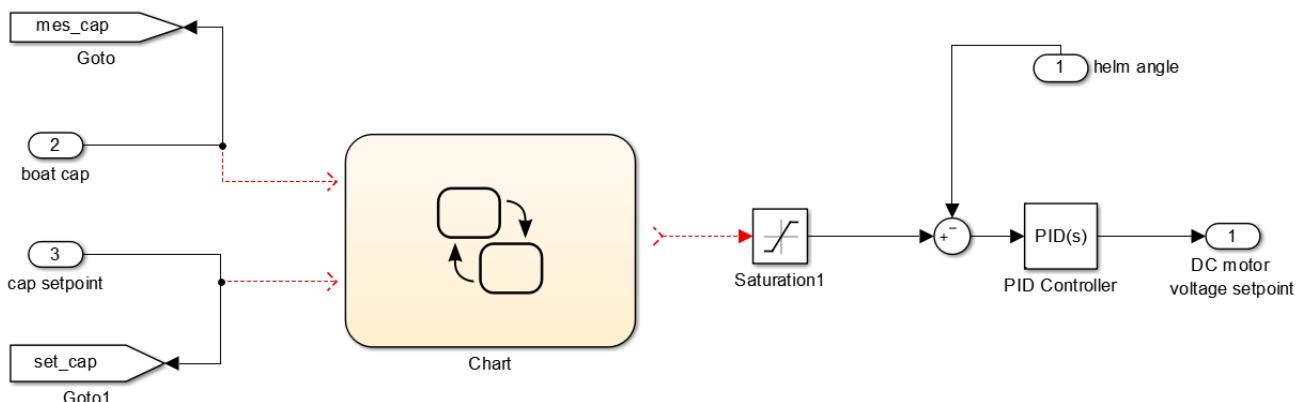


Figure 279: positioning a chart in a Simulink model

We can see that the “chart” currently has no input/output. They will be defined later.

C. Creating an elementary state diagram

Double-click on the “chart” to open the **Stateflow** environment.
The **Stateflow** toolbar will appear on the left of the window with controls that will enable the creation of a state and a “default transition”.

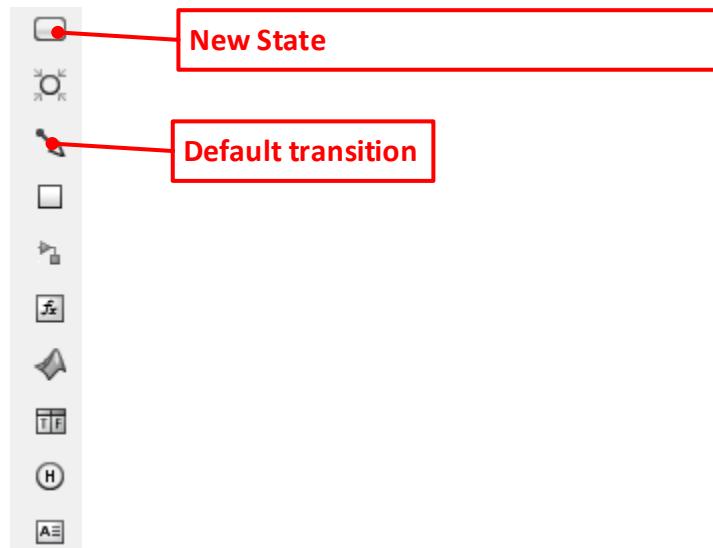


Figure 280: commands for creating a state and a “default transition”

1. Creating states

It is now possible to create states to program the system logic.

Using the new state creation command, place and name the three states following Figure 281. To do this, **left-click** on the new state creation command and drag the state into the working window.

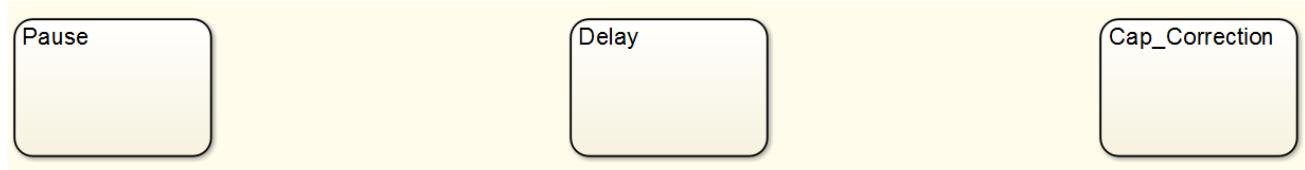


Figure 281: creating system states

In this example, we'll begin by working with **exclusive** states, where only a single state can be active at each time step. We will then see that it is also possible to create **parallel** states which can be active simultaneously.

2. Creating a default transition

Place a default transition on the “Pause” state. The default transition enables a state to be activated at the beginning of the simulation. The presence of a **default transition** in the diagram is **essential**.

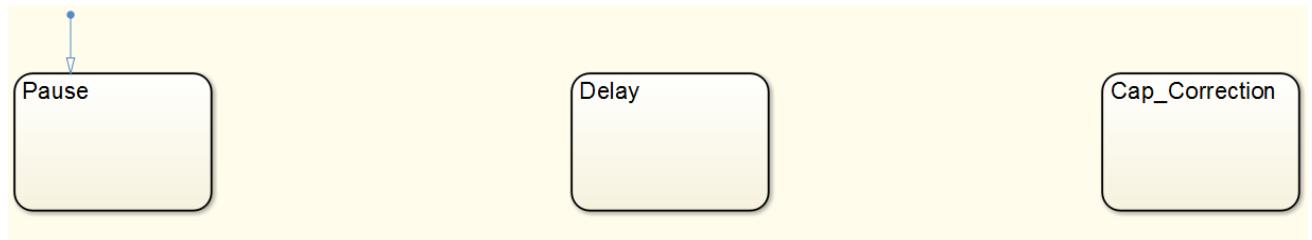


Figure 282: creating a default transition

3. Creating transitions

To create a transition, move the mouse cursor to the edge of the state at the start of the transition. When the cursor becomes a cross shape, left-click and drag the cursor to the end state of the transition.

Create transitions to obtain the configuration of Figure 283.

The transitions are equipped with invisible points that enable them to be shaped into the desired form. Left-click and drag the mouse to change their shape.

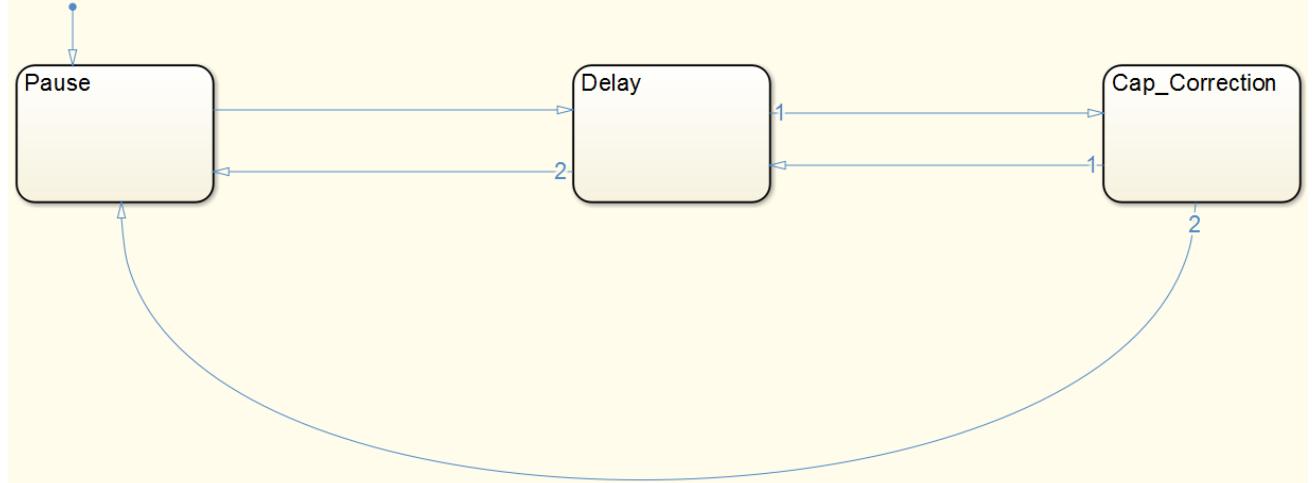


Figure 283: creating transitions between states

4. Creating actions in states

We now need to assign each state actions to use on the system.

To do this, we need to use an action keyword which will determine the time at which the action will be executed.

The normal keywords are:

- **entry** (or **en**): the action will be executed only upon activation of the state
- **exit** (or **ex**): the action will be executed only upon the deactivation of the state
- **during** (or **du**): the action will be executed at each time step when the state is active.

Complete the actions for the states following Figure 284.

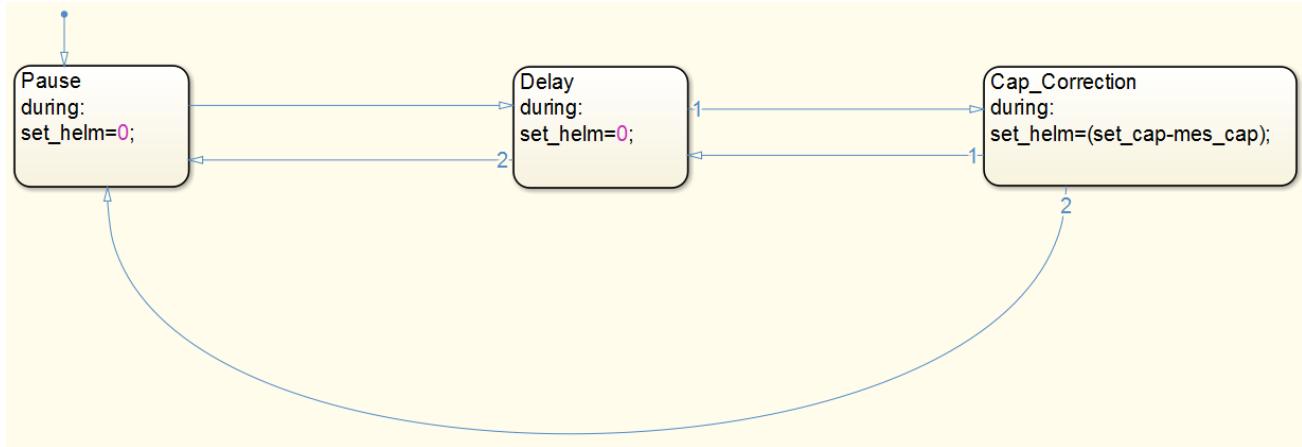


Figure 284: attaching actions to states

For the “**Pause**” and “**Delay**” states, using the **entry** keyword only requires the action `set_cap=0` on activation of the state. The variable `set_helm` remains at 0 and does not vary when these states are active. On the other hand, when the system is in the “**Cap_Correction**” state, the `set_helm` variable must be continually modified as the course of the boat changes, thus requiring the use of the keyword **during**.

5. Creating transition labels

We now need to define the conditions which will enable the change from one state to another. Transition labels represent logic conditions which must be verified in order to change from one state to another.

There can be different types of information present in the transition label.

- A comment: `/*comment*/`
- A condition: `[condition]`

To create a transition label, left-click on the transition, then left-click again on the question mark that appears. You can then input the transition label. To move a transition label, drag it using the left button on the mouse.

Create transition labels following Figure 285.

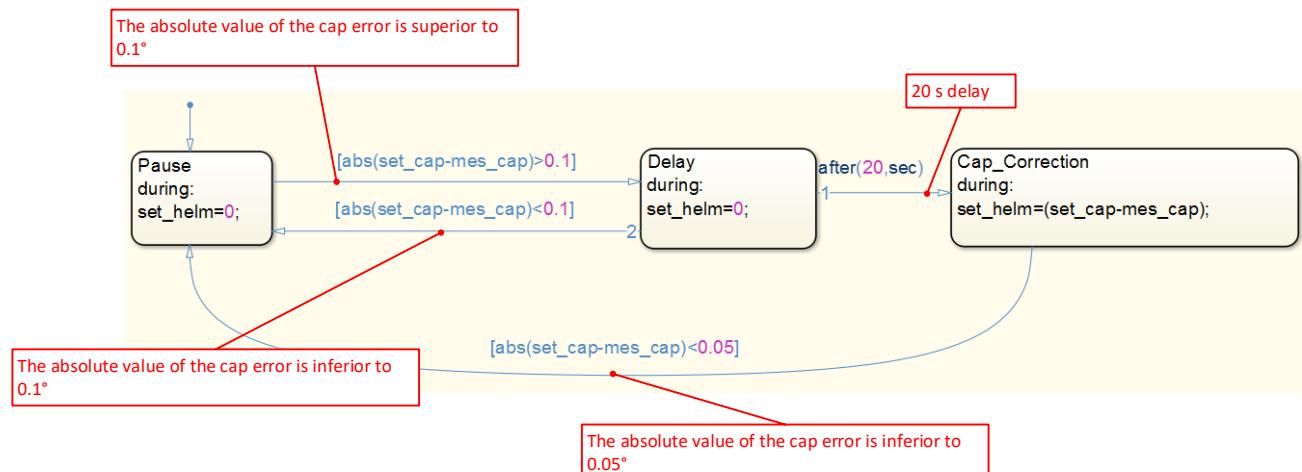


Figure 285: writing transition labels in a states diagram

Transition labels also offer other possibilities which are described on page 266.

6. Definitions of input and output variables of the state diagram

If we return to the subsystem representing the information chain, we can see that the chart has no input or output connecting it with the system.

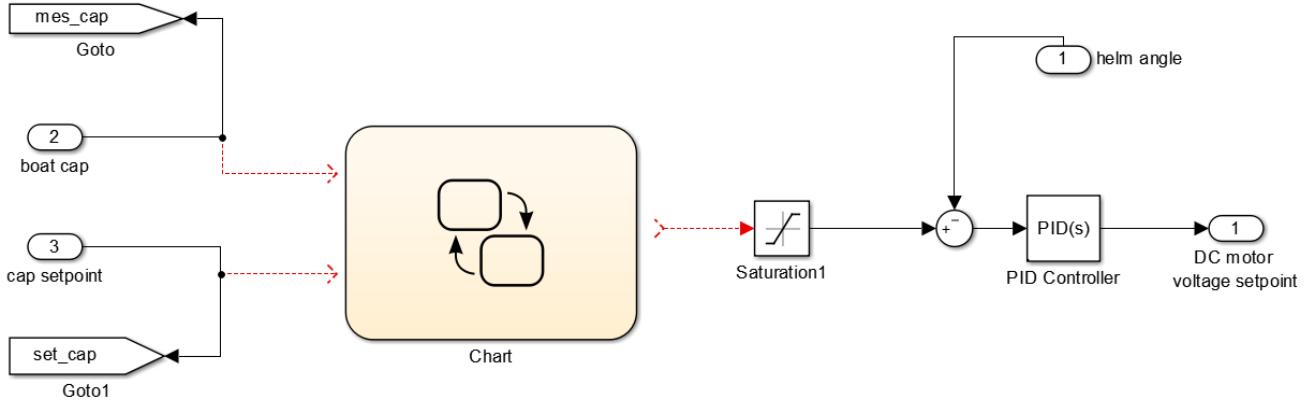


Figure 286: chart not connected with the model

There are several ways to create the input and output variables. The most simple is to launch the simulation. **Stateflow** will automatically make suggestions for input and output data for the chart in accordance with the logic to be programmed.

Launch the simulation of the model.

An error message will appear indicating that the simulation is impossible and the **Symbol Wizard** window will open and suggest default variables.

Check that the default suggestions correspond with the control logic for the system. If necessary, you can modify them using the drop-down menus.

Tick the View created data/events in Model Explorer box, then click OK.

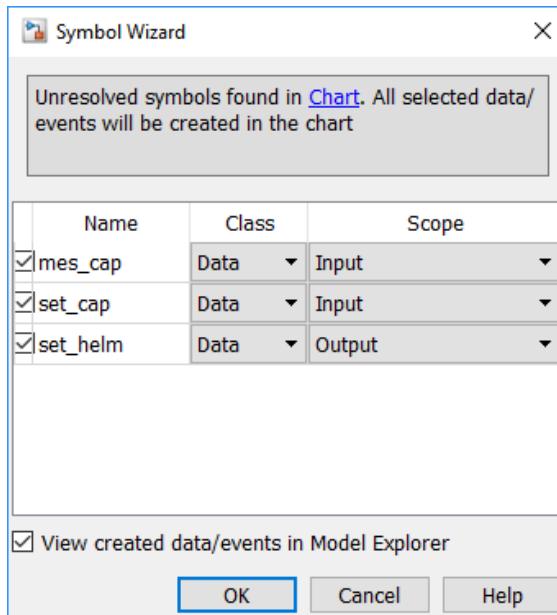


Figure 287: adding variables to the diagram using Symbol Wizard

On closing the **Symbol Wizard** window, the **Model Explorer** window will open. The **Model Explorer** enables all of the variables that were created for the model to be displayed. The tree hierarchy on the left of the window can be used to navigate the system.

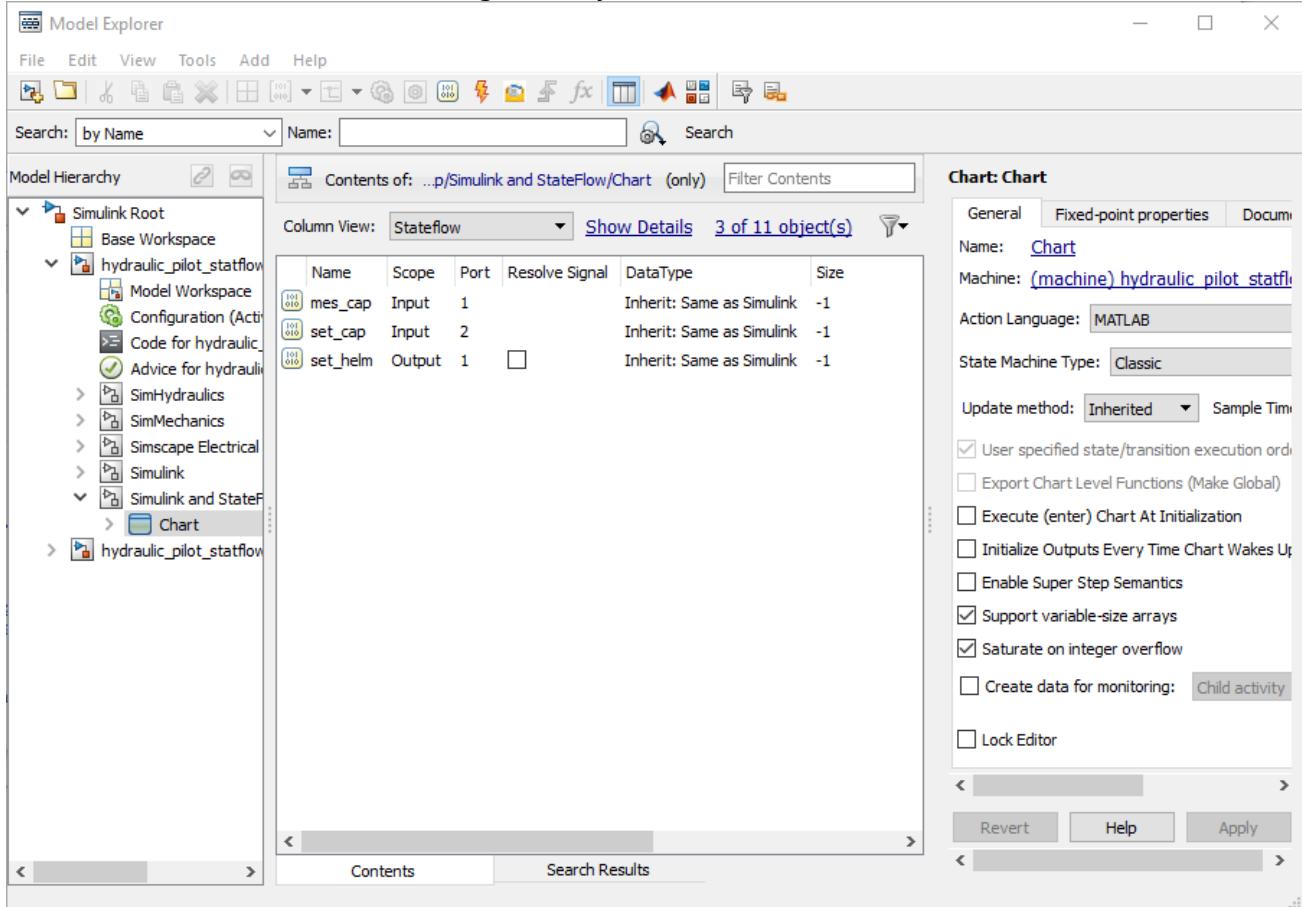


Figure 288: displaying variables in Model Explorer

It is also possible to modify the characteristics of the different variables. You can return to **Model Explorer** at any time to modify the characteristics of a variable by clicking on in the main tool bar.

It is also possible to add variables manually using the tool bar with the **Chart/Add Inputs & Outputs/Data Inputs from Simulink** command to add an input and the **Chart/Add Inputs & Outputs/Data Outputs from Simulink** commands to add an output.

Return to the sub-system representing the information chain to display the connections that appear on the Chart.

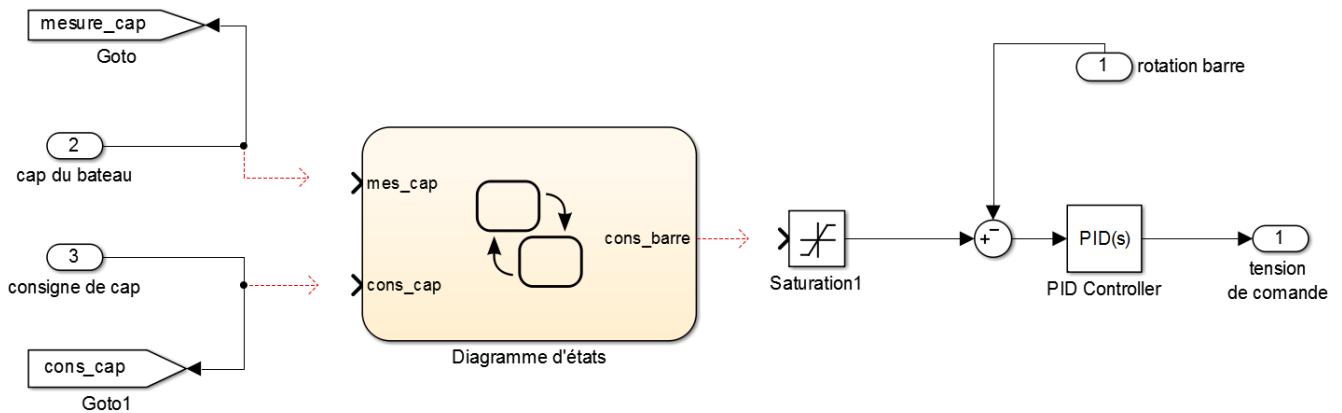


Figure 289: chart before connection with the system

The connections having been created automatically and they can also be moved around on the chart. Check that the reference course for the course is correctly connected to cons_cap and that the course measurement is correctly connected to mes_cap.

If the connections are inverted, open **Model Explorer** and input the correct port numbers as indicated by Figure 290.

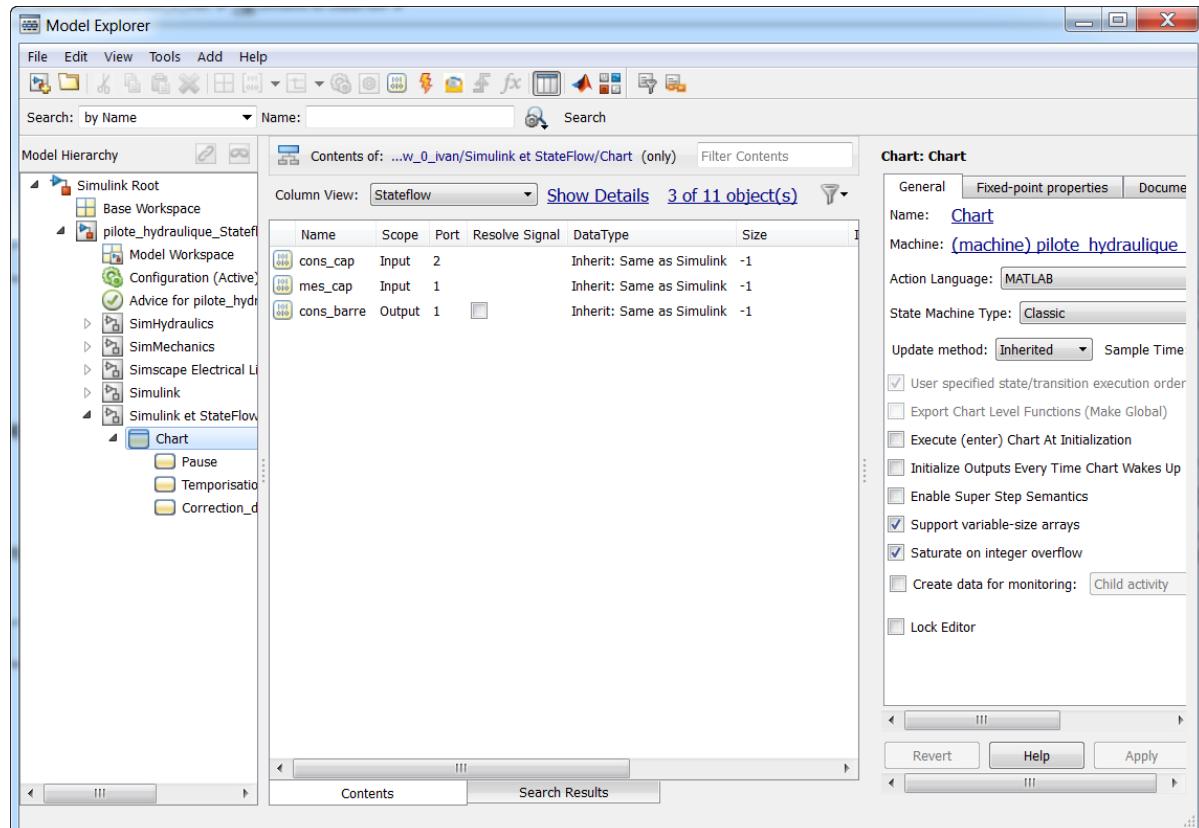


Figure 290: modifying the port numbers of a chart

Connect the chart with the Simulink block diagram following Figure 291.

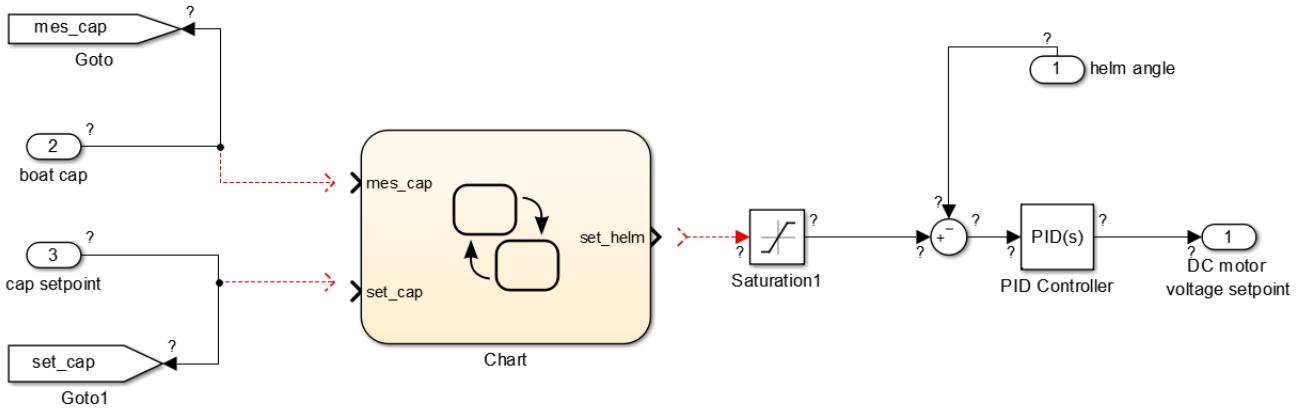


Figure 291: connecting the chart with the Simulink block diagram

7. Simulating a states diagram

Launch the simulation and observe the monitoring of the course by displaying the scope that shows the reference course and the actual course followed by the boat.

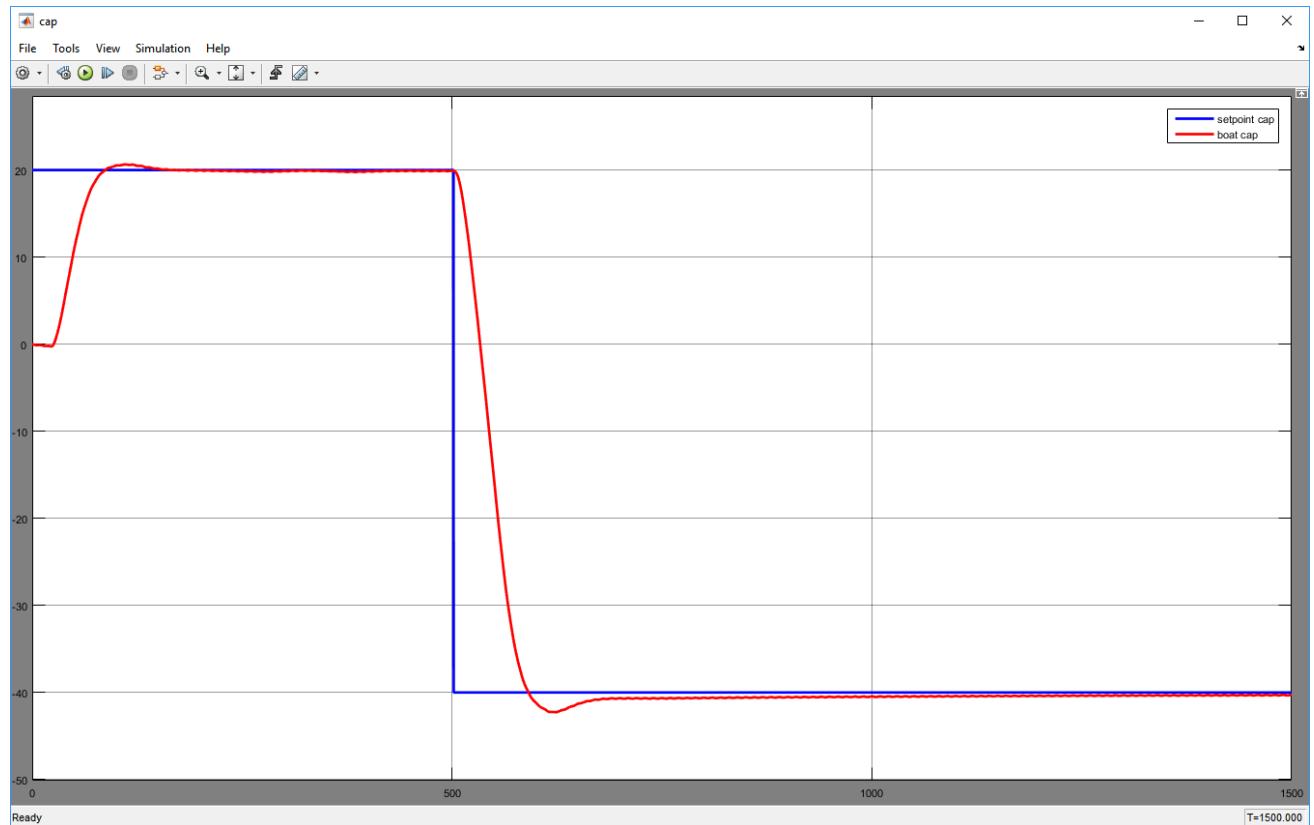


Figure 292: simulation results

D. Architecture of state machines

1. States hierarchy

The state diagram created previously had 3 states of the same level. No hierarchy is present between these states. In order to be able to model more complex logic behavior, **Stateflow** uses the “**superstate**” concept.

A superstate is a state that can contain other states of lower hierarchical levels, called **substates**.

- **Superstate (or outer state)**: parent state containing other states
- **Substates (or child state)**: child states contained in a outer state. The substates can only be active if the outer state is active.

Using superstates and substates improves the readability of the model. There is no limit to the number of hierarchy levels that can be constructed in a diagram.

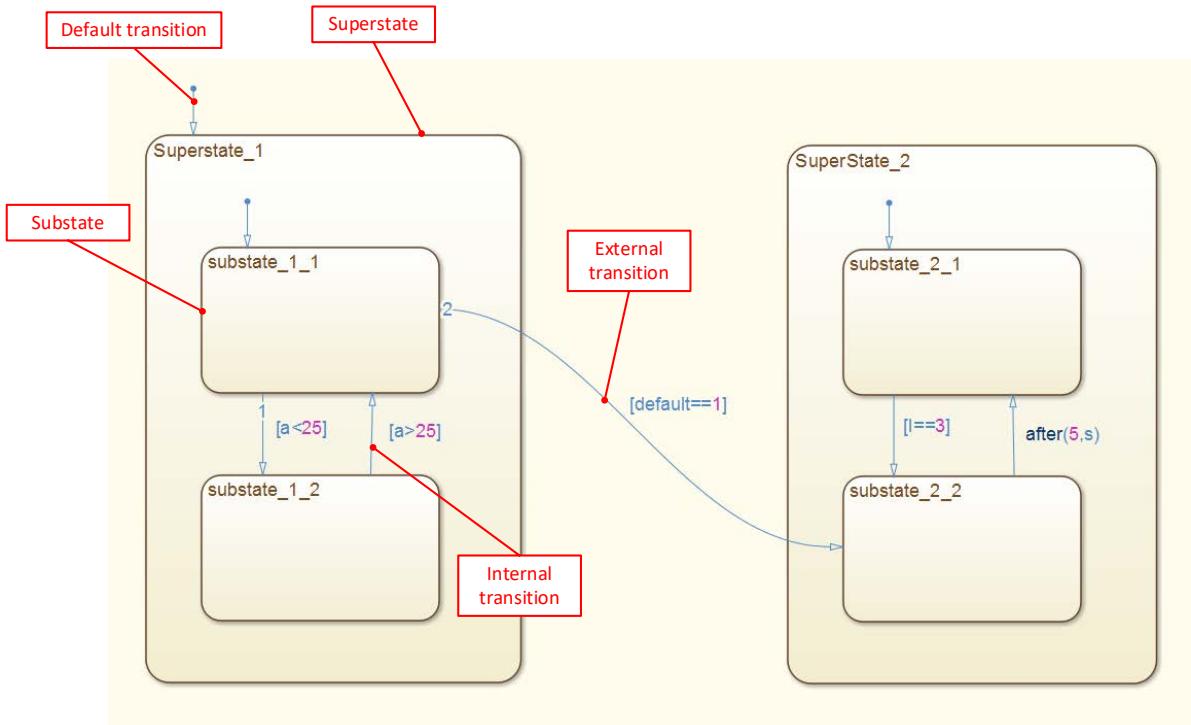


Figure 293: example of superstate and substate

2. Test priorities for transitions

These more complex architectures require details regarding change rules and test priorities for the various transitions.

These transitions can be defined as:

- **internal transitions**: a transition that does not cross the edge of the state or is surrounded by the state
- **external transitions**: a transitions which crosses the border of the state

Test rules for transitions:

- superstate transitions are tested before those of substates
- external transitions are tested before internal transitions

3. Parallel states

At each level of the hierarchy, the sub-states can be **parallel** or **exclusive**.

- **If the substate is exclusive:** a single and unique substate cannot be active within the same hierarchical level (on the condition that the parent superstate is active)
- **If the state is parallel:** all the substates on the same hierarchical level are active simultaneously (on the condition that the parent superstate is active). It is not possible to define a transition to or from a parallel state, as its activation will be exclusively conditional on the activation of its parent state.

In **Stateflow**, the parallel states are displayed with dotted edges.

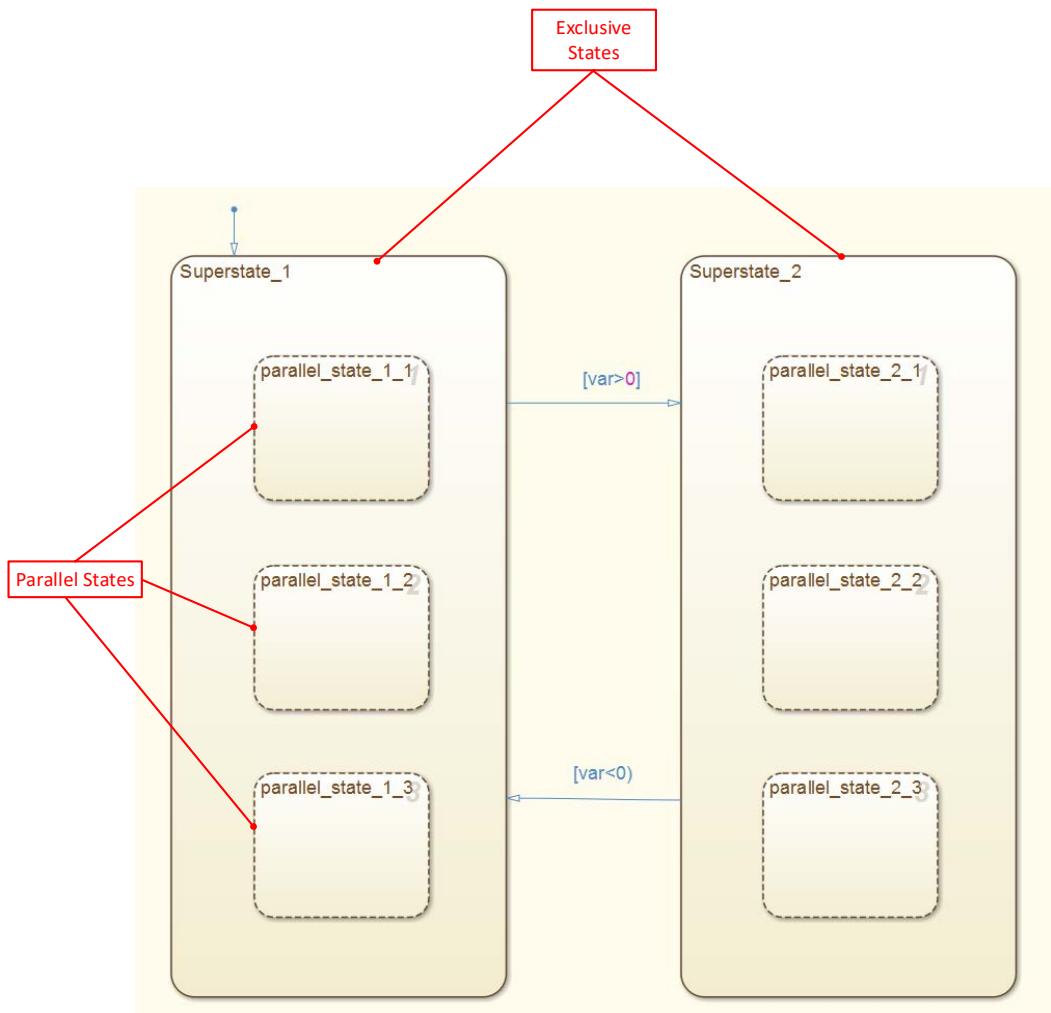


Figure 294: view of parallel and exclusive states

E. Adding hierarchical levels and parallel states into a state diagram

We are now going to return to the state diagram for the course control of the hydraulic pilot of a boat and add new functionalities.

When the pilot is in “Cap_Correction” mode, a red light should light up to alert users that the automatic pilot is using energy.

The switching on of this light will be managed by the state “Light control”. A “red light” variable will be at 1 when the light is on, and at 0 when the light is off.

Create the superstates “Course monitoring”, “Light control” and “Main” while maintaining the hierarchy shown by Figure 295. The procedure for creating a superstate is the same as for creating a state. Care should be taken to ensure that the edges of the parent state surround the child states. **Stateflow** automatically takes into account the hierarchy between the states based on their position in the graphic window. If a conflict prevents **Stateflow** from establishing a non-ambiguous hierarchy between the states, the states which are causing problems will appear in red.

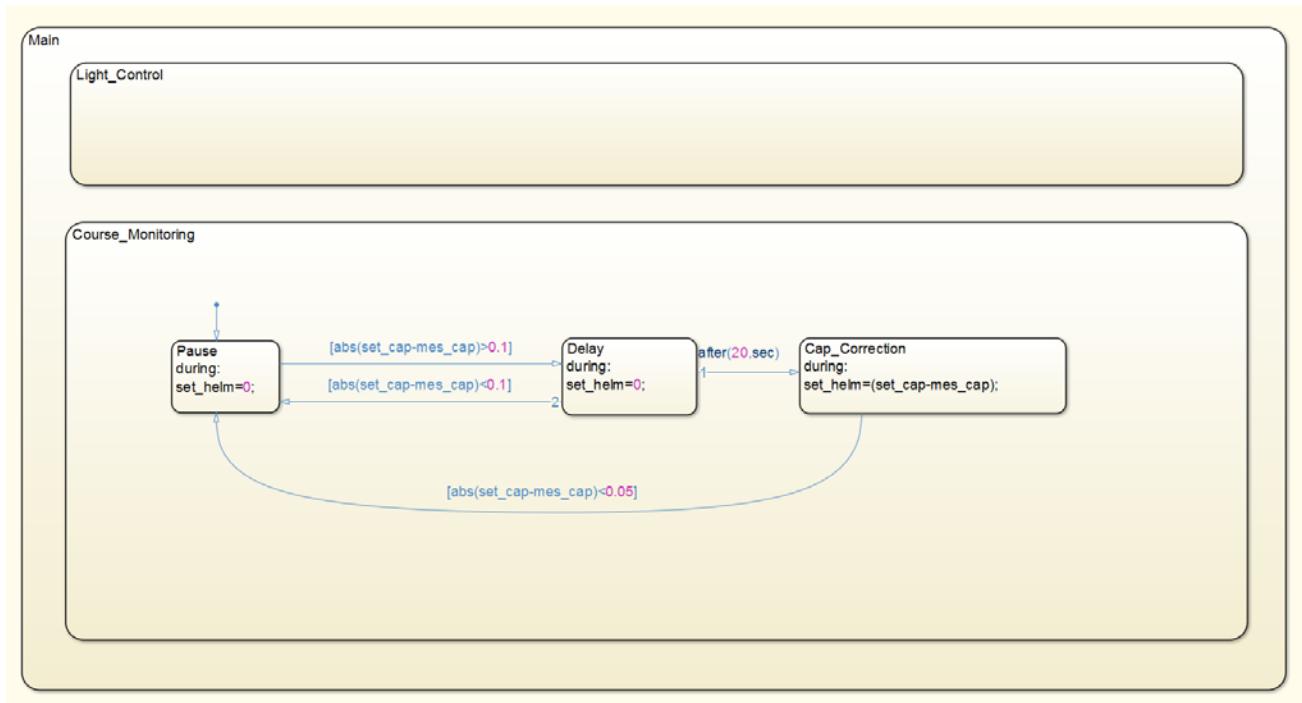


Figure 295: creating superstates

The “Main” state has 2 child states:

- “Course_Monitoring”
- “Light_Control”

These child states must be **parallel** states as the light should be lit up when the “>Course_Monitoring” state is active. If the command system is working correctly, it causes the two child states to be active simultaneously when the “Main” state is activated.

The “Course_Monitoring” state has 3 child states:

- “Pause”
- “Delay”
- “Cap_Corection”

These child states must be **exclusive** states, as these states must be activated successively when the parent state “Course_Monitoring” is activated. Only a single one of these states can be active at once.

To indicate to **Stateflow** that the child states of the parent state “Course_Monitoring” are parallel states, right-click inside the “Course_Monitoring” state, but outside of the child states (Figure 296). Choose **Decomposition/parallel** from the contextual menu.

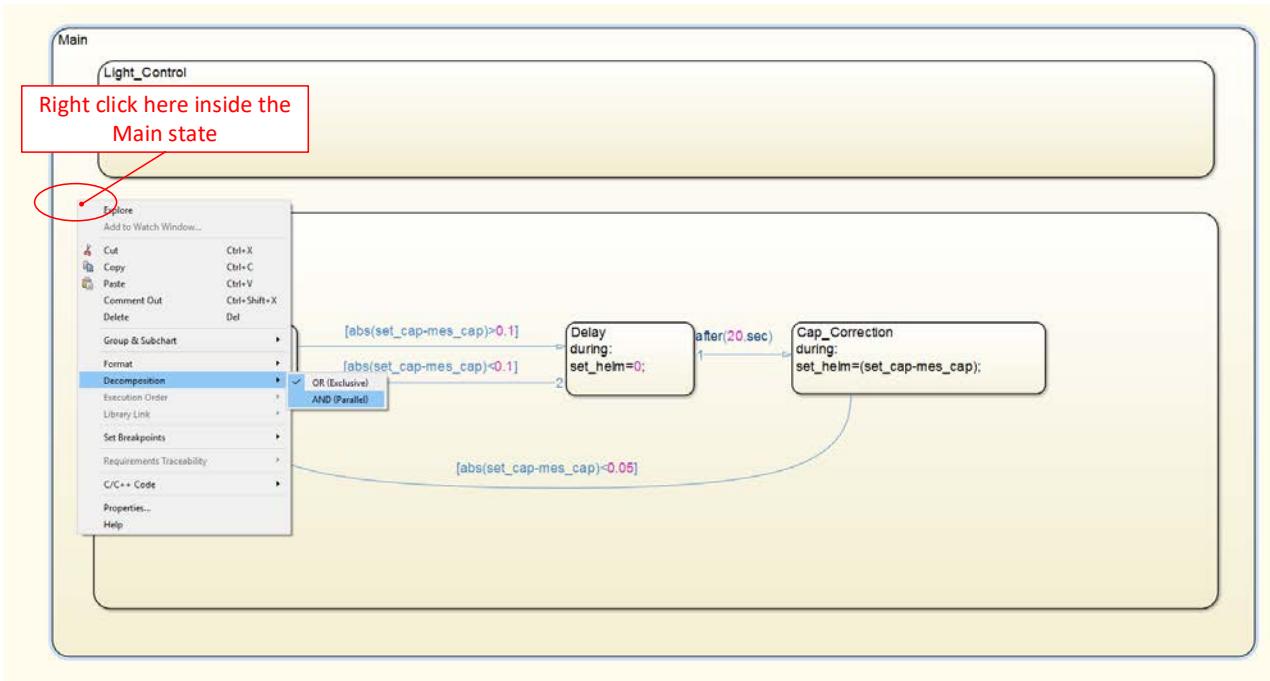


Figure 296: creating parallel substates

The parallel substates will now appear with dotted lines.

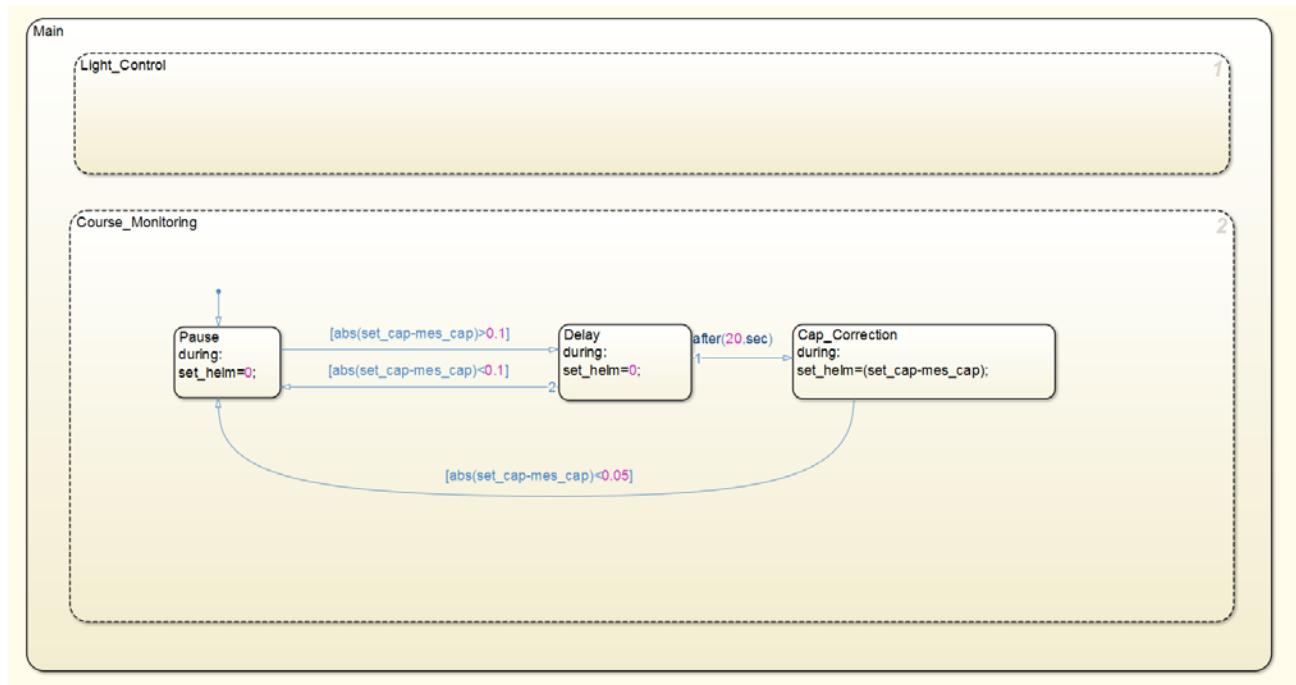


Figure 297: parallel states in Stateflow

We now need to indicate that the “red_light” variable will be at 1 when the “**Cap_Correction**” state is active.

Stateflow has a variety of internal variables to do this which define the activation of states.

The command `in(state_name)` returns the value 1 when the state is active and the value 0 when the state is inactive.

The name of a state is composed of the entire hierarchy tree of the parent states.

Modify the state “**Light_Control**” as indicated in Figure 298.



Figure 298: using internal variables to evaluate the activation of a state

Use the **Chart/Add Inputs & Outputs/Data Outputs to Simulink** command to create the **red_light** output variable.

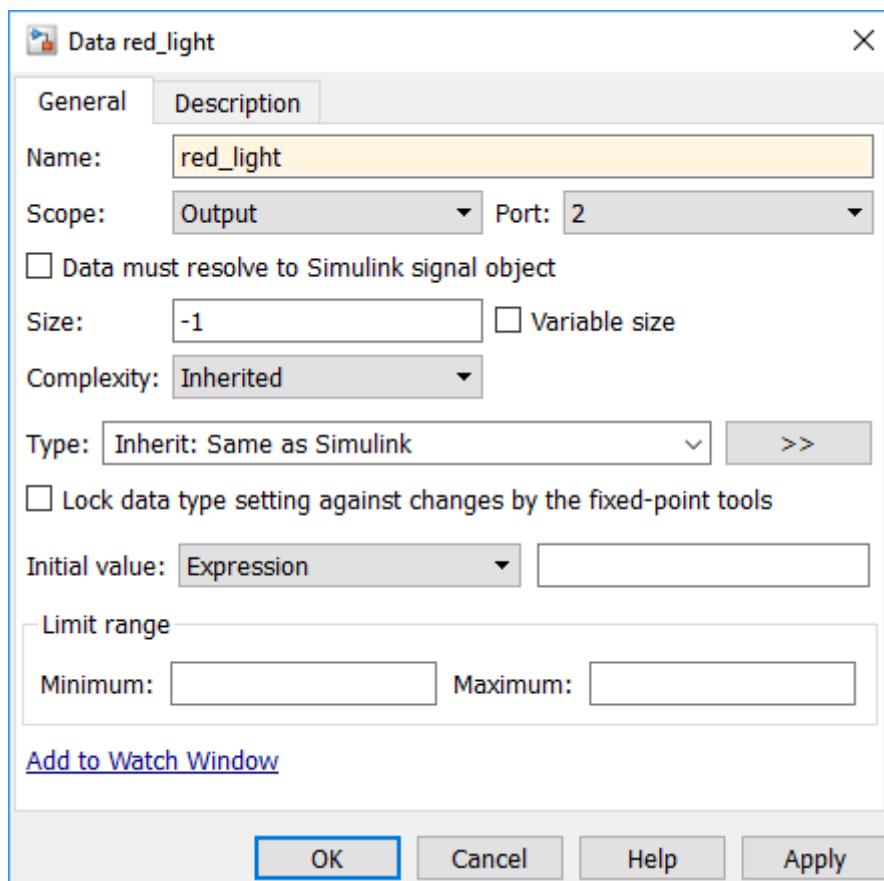


Figure 299: creating an output variable to Simulink

Before launching the simulation, we need to add a “default transition” to the “Commande_de_cap” (Course control) state to activate the diagram on startup.

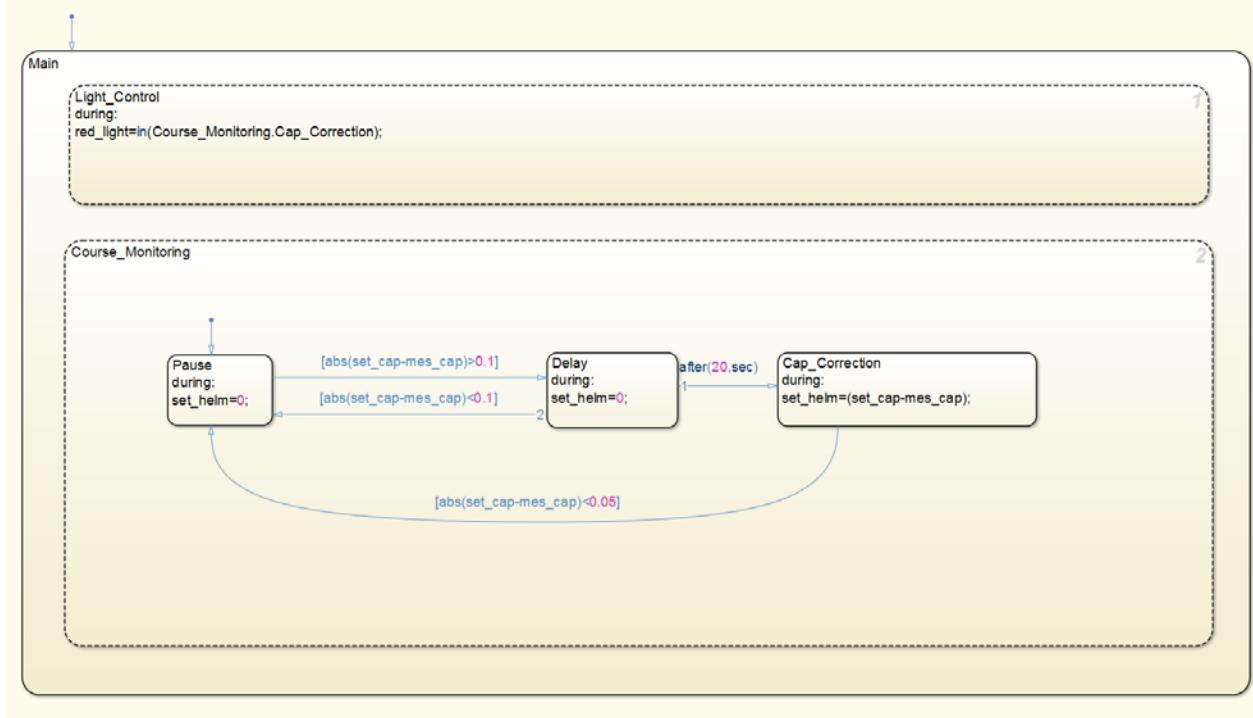


Figure 300: completed course control chart with superstates and parallel states

Connect the “red_light” output variable to a scope and to a “lamp” from the dashboard library.

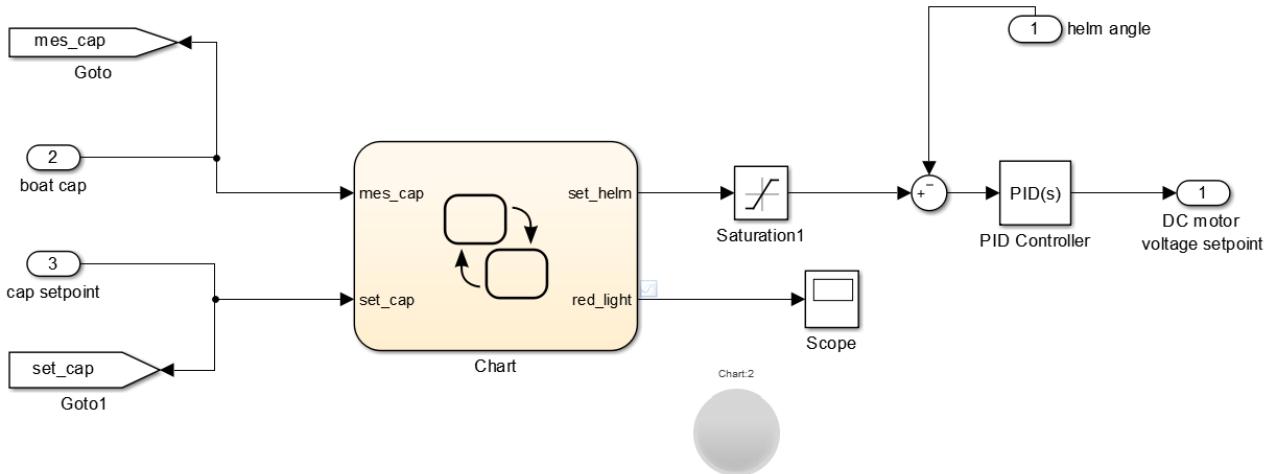


Figure 301: connect the chart output variable to a scope

If necessary, the complete model is available in the file **hydraulic_pilot_Stateflow_final_US.slx**.

Launch the simulation and observe the changes to the state of the red light variable in the scope and the light of the lamp.

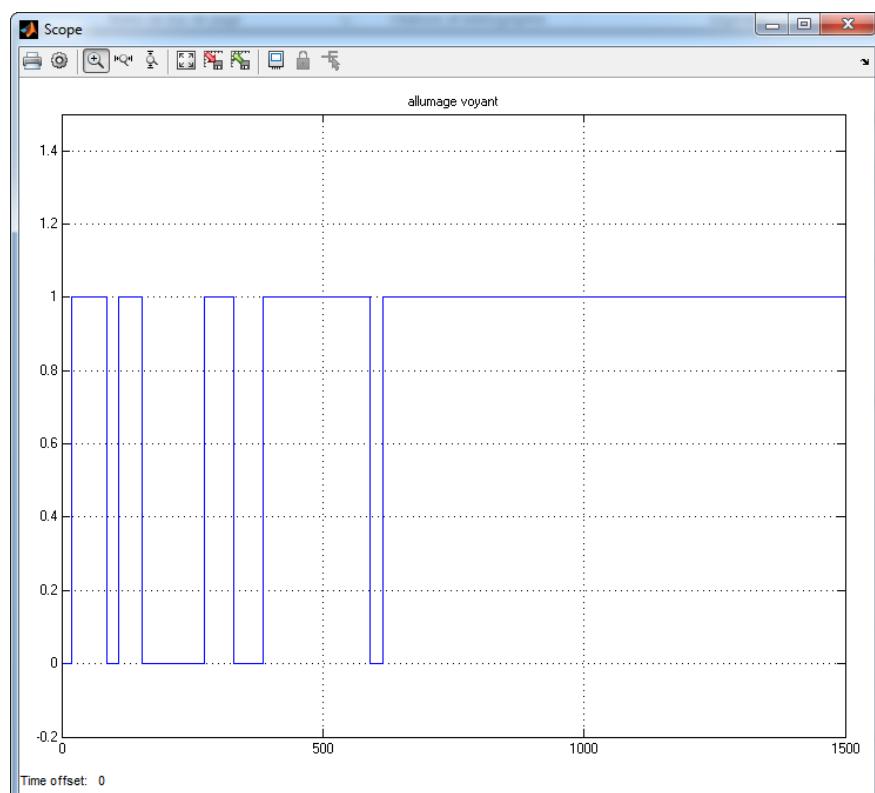
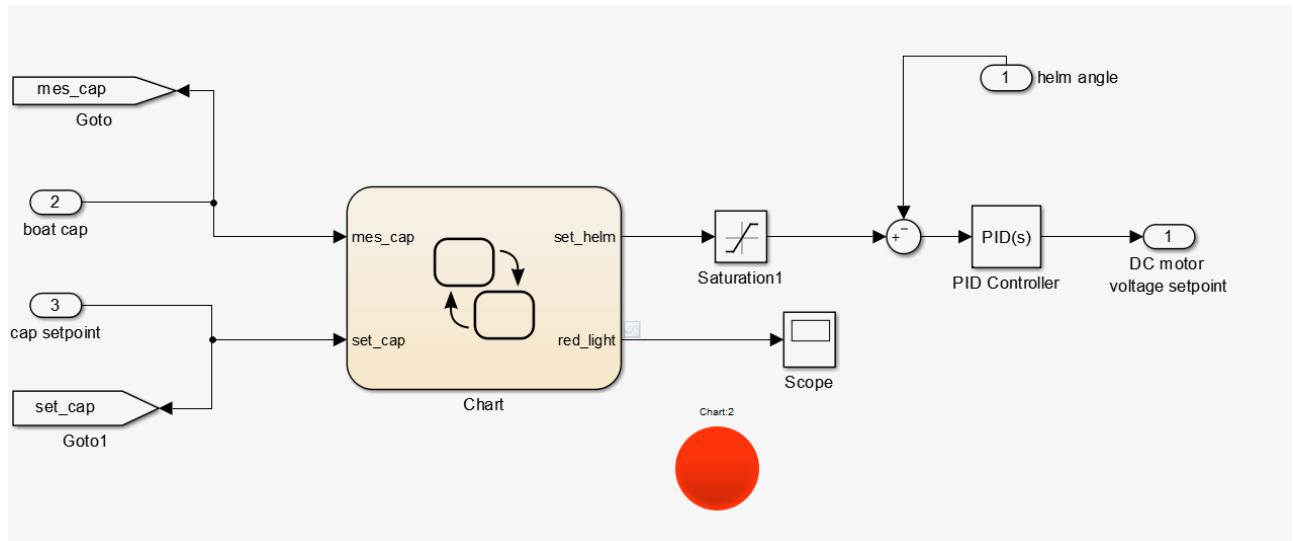
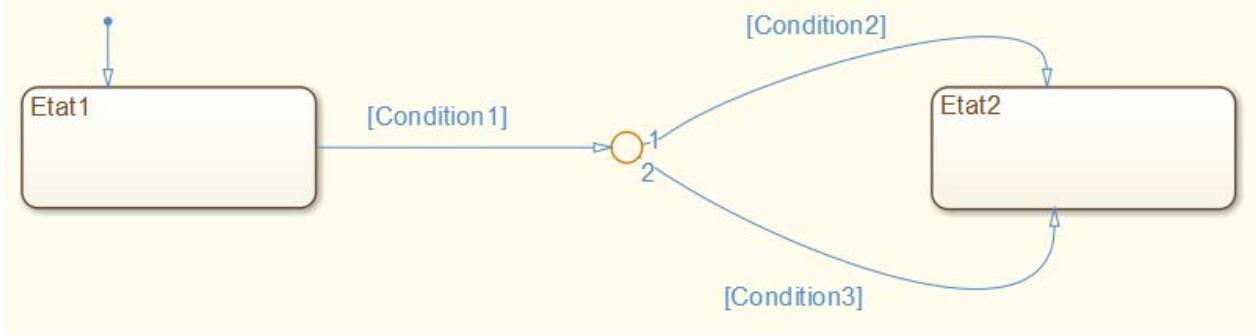


Figure 302: displaying the activation state of the light

F. Summary and additional useful Stateflow commands

Useful commands	
Functions	Commands/syntax
<p><i>Creating a state:</i></p> <div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> Nom /*commentaires*/ mot clé: actions </div>	<p><i>Keywords:</i></p> <p>en or entry: the action will be carried out only upon activation of the state</p> <p>du or during: the action will be carried out continuously as long as the state is active</p> <p>ex or exit: the action will be carried out only upon deactivation of the state</p>

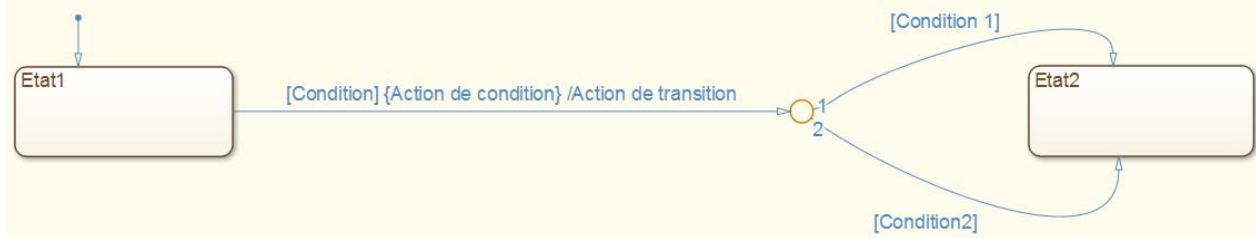
Creating a transition with a junction:



To create a transition with multiple paths, it is possible to use a junction  available in the “Chart” menu.

To continue to State2, **[Condition1]** must be true and **[Condition2]** or **[Condition3]** must be true. The path numbered **1** is evaluated before the path numbered **2**.

Creating a transition that includes actions:



The **Condition** enables the change from State1 to State2 to be evaluated.

To continue to State2, **[Condition]** must be true and **[Condition1]** or **[Condition2]** must be true.

A **Conditional action** is an action that is executed only if the condition test result is true, even if the transition path does not result in the activation of State2.

If **[condition]** is true the **{conditional action}** will be executed even if **[Condition1]** and **[Condition2]** are false.

A **Conditional action** is an action that is executed only if the condition test result is true, even if the transition path does not result in the activation of State2.

/ **Conditional action** is executed during the change from State1 to State2.

<i>Testing of variables in transitions:</i>	a and b a or b in addition to a a equal to b	a&&b a b !a a==b
---	---	----------------------------

Monitoring the activation of states:

in(State): this internal variable is true (value 1) as long as the state is active.

<i>Delay of 15s in a transition</i>	after(15,s)
-------------------------------------	-------------

Chapter 7: SimMechanics management

I. Introduction to SimMechanics

SimMechanics is a MATLAB tool that enables a 3D CAD model of the system imported from Solidworks (or any other CAD software) to be incorporated into a multi-physical model. This enables the movements of the solids during the simulation to be visualized, and allows us to take into account the dynamic behavior of moving solids and all the geometric non-linearities.

The results from the multi-physical model are closer to the results observed in the actual system by considering these new parameters without the need to write equations to define the dynamic behavior of the system.

There are two generations of the SimMechanics tool, 1G and 2G. This document relates to the use of the second generation (2G) SimMechanics, which has been greatly improved in comparison to SimMechanics 1G.

A. Analyzing a SimMechanics 2G model

In SimMechanics, systems are represented in the form of a bond graph. The solids are thus represented by blocks connected to each other with bonds.

Open the file "**hydraulic_pilot_SimMechanics_2G_US.slx**"

This file was imported directly from Solidworks. Methods for importing files will be covered in later chapters.

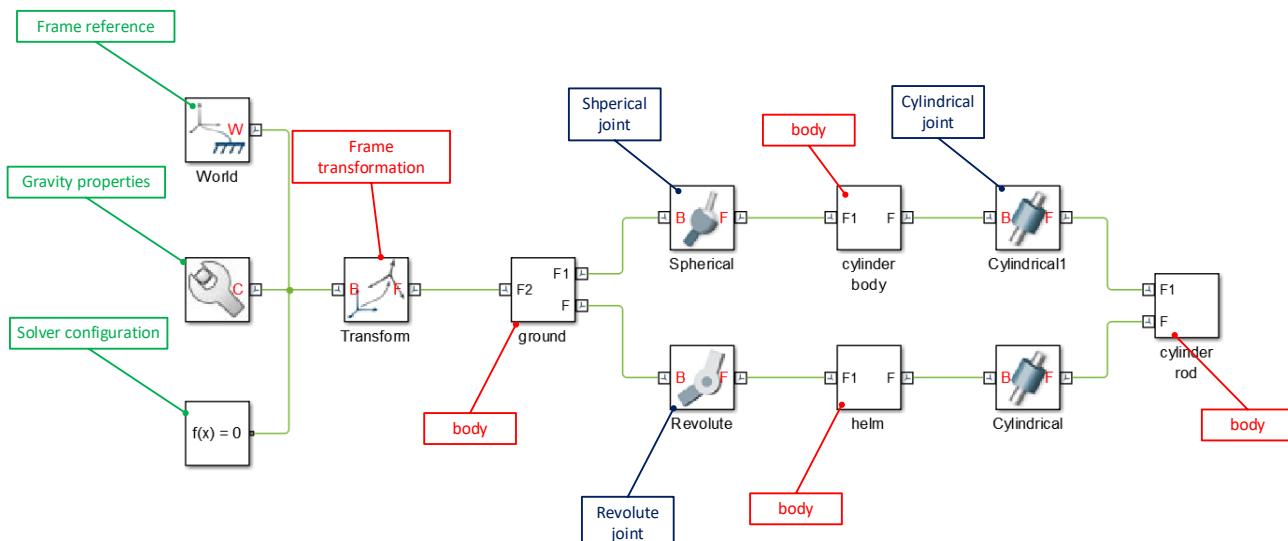


Figure 303: SimMechanics 2G model of the mechanical part of the hydraulic pilot

To improve the readability of the model, it is recommended that masks are created for the “solid” blocks to directly display their shapes.

Open the file “*hydraulic_pilot_SimMechanics_mask_0_US.slx*”

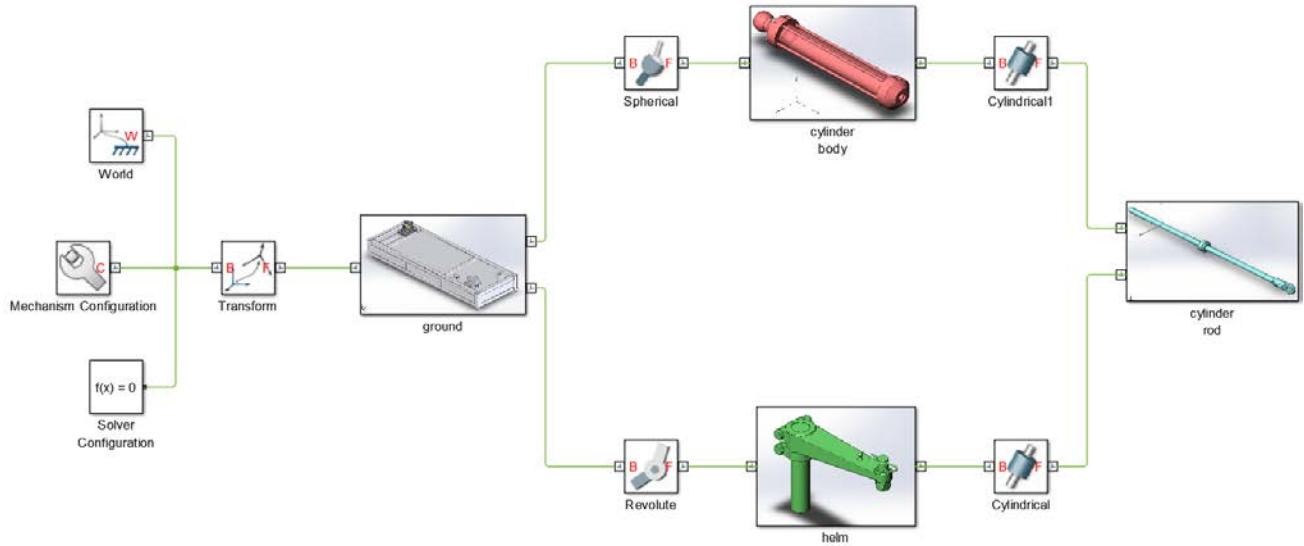


Figure 304: SimMechanics 2G model of the hydraulic pilot with masks over the solids

Launch the simulation of the model.

The **Mechanics Explorer** window will appear and the 3D model of the mechanical section of the hydraulic pilot can be displayed.

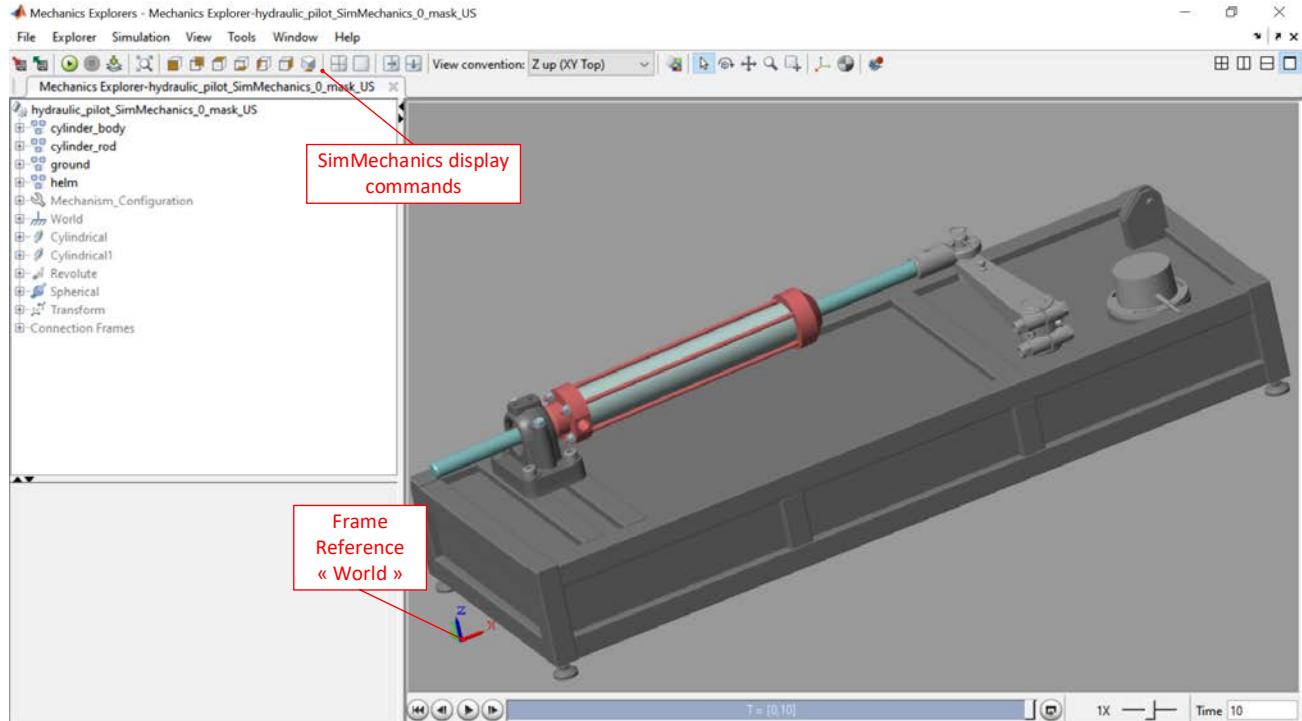


Figure 305: Mechanics Explorer window showing the mechanical section of the hydraulic pilot

The SimMechanics toolbar offers a range of different display options.

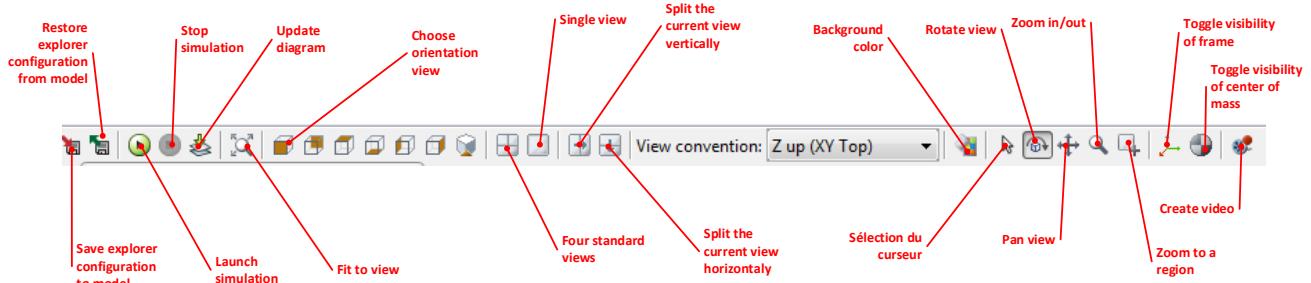


Figure 306: SimMechanics visualization bar options

Note that no movement is observed if there are no connections being piloted.

B. Configuring gravity

Return to the model window and open the Mechanism Configuration block

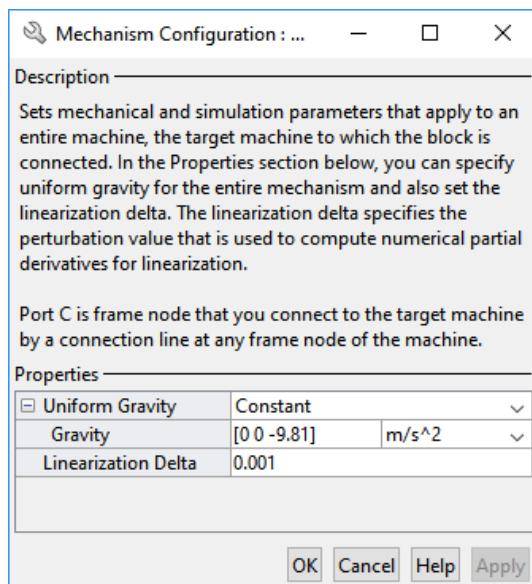


Figure 307: setting gravity in SimMechanics

Gravity is located on the Z axis by default (value -9.81 m/s^2), which corresponds to the actual functioning of the system. The reference point used to orient the gravity is the fixed "World" reference point.

To see the influence of gravity on the simulation, we are going to place it on the X axis.

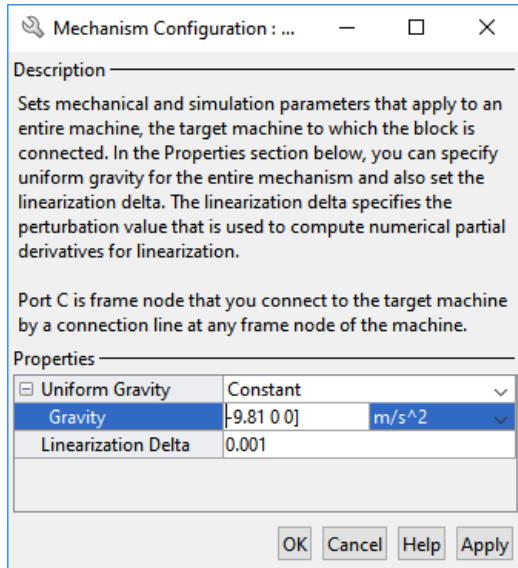


Figure 308: modifying the action of gravity

Relaunch the simulation and watch the movement of the system. We can see an in-out movement with the actuator rod, which corresponds to the action caused by gravity. With friction not being modelled, the movement is periodic.

Reset the gravity to the Z axis to return to the actual operating conditions.

II. Integrating a SimMechanics model into a multi-physics model

In this section we are going to learn how to integrate a SimMechanics model into a multi-physics model. To do this, we will re-use the model of the boat's hydraulic pilot.

Open the file **hydraulic_pilot_SimMechanics_start_US.slx**

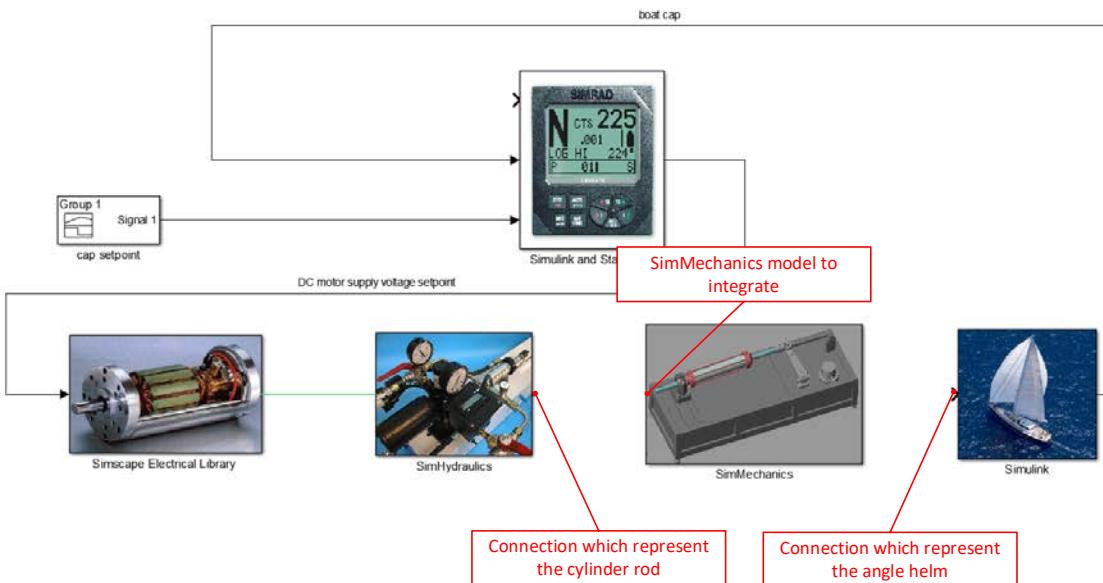


Figure 309: hydraulic pilot model with SimMechanics model to be integrated

The **SimMechanics** sub-system contains the model of the mechanical section of the pilot but it is not connected to the rest of the model.

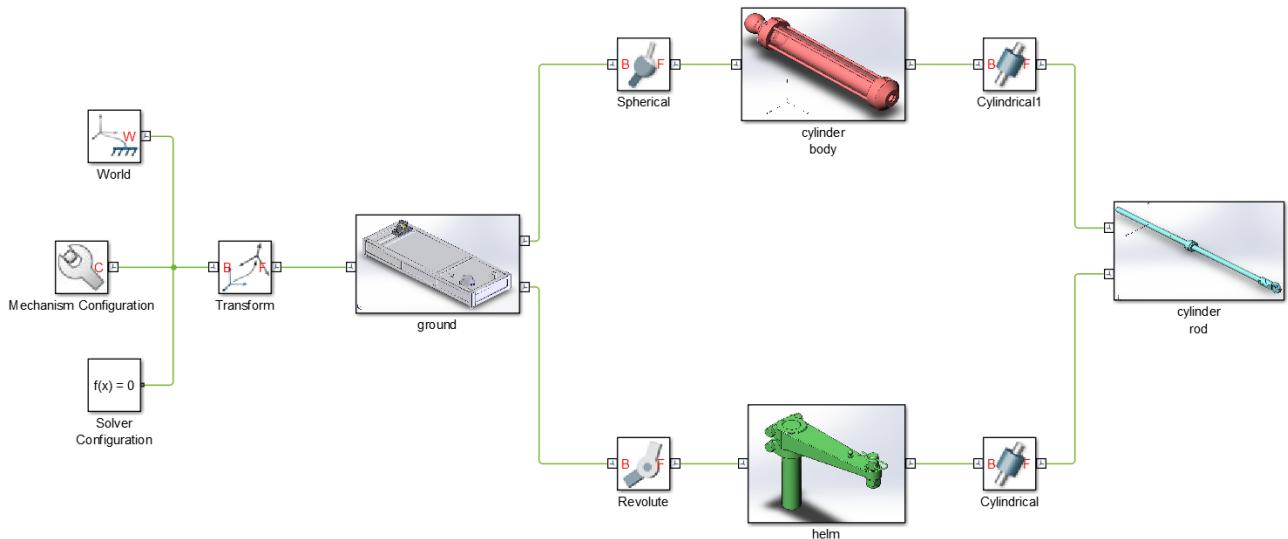


Figure 310: view of the SimMechanics system not connected to the rest of the model

In Figure 309 it is possible to view the connections that the **SimMechanics** sub-system needs to have with the rest of the model.

- On input, the model receives a physical connection that represents the actuator rod. This is a **Physical Conserving Port** type port (Simscape)
- On output, the **SimMechanics** sub-system should return the helm angle in order to act on the boat and give the helm angle to the information chain as input information.

A. Model connections

In order to connect the model we need to create two ports inside the **SimMechanics** sub-system:

- A Simscape port to connect to the actuator rod
- A Simulink port to return the rotation angle of the helm

Designation	View	Library
Simscape connections port	 Connection Port	Simscape/Utilities
Simulink connections port		Simulink/Ports & Subsystems

Place the ports and rename them as in Figure 311.

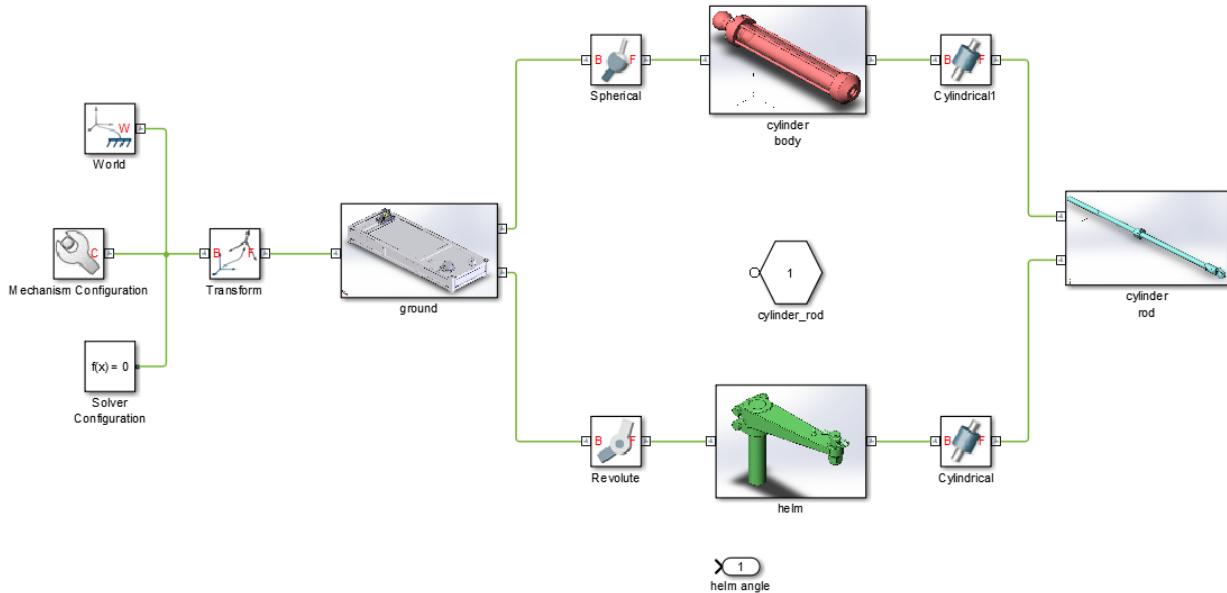


Figure 311: placing the connection ports in the model

Ports will appear on the SimMechanics sub-system. **Connect** these ports to the model.

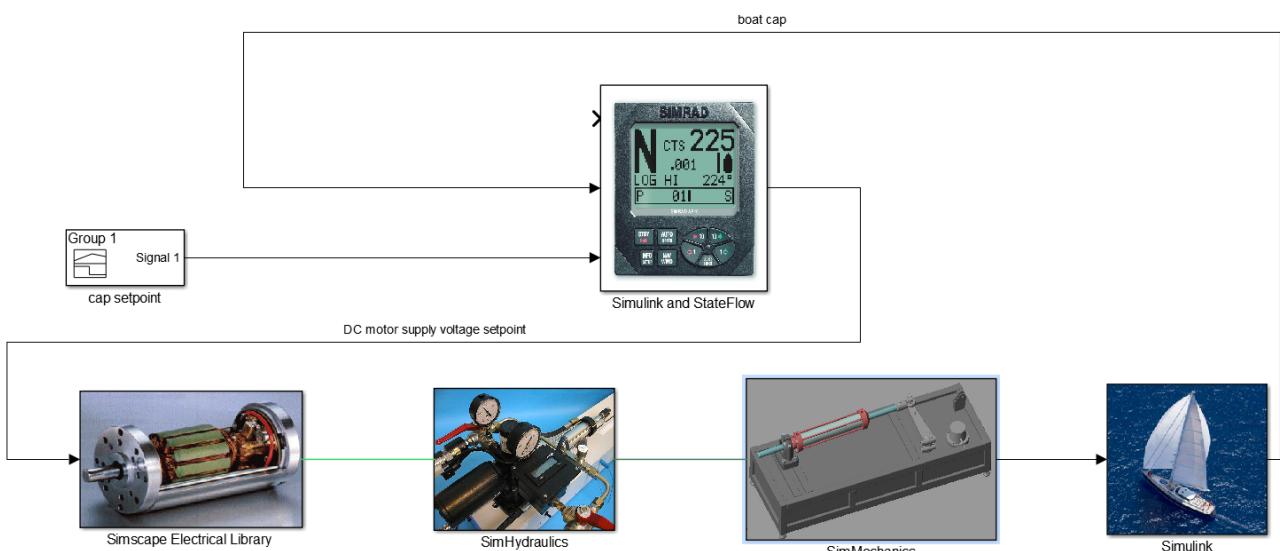


Figure 312: connecting the sub-system ports to the model

B. Interfacing between Simscape and SimMechanics

In this example, the “cylinder rod” connection originating from **Simscape** must serve to activate the connection between the cylinder body and the rod. In **SimMechanics 2G**, the connections are activated by “Force” or “Torque” (moment) type forces. The sliding pivot connection between the cylinder body and the actuator rod must therefore be activated through a force on the cylinder rod using a force sensor. This force will act on the mechanical part of the system. This causes the entire mechanism to move, taking into account the dynamic of the mechanical section, which results in the output speed of

the actuator rod. In order to avoid breaking the physical loop and to enable the actuator rod modelled in **Simscape** to move at the same speed as the actuator rod in the **SimMechanics** model, it is important to reinject this speed into the **Simscape** model. This ensures an equivalence of behavior between **Simscape** and **SimMechanics** throughout the operation of the mechanism.

1. Translational interface between Simscape and SimMechanics

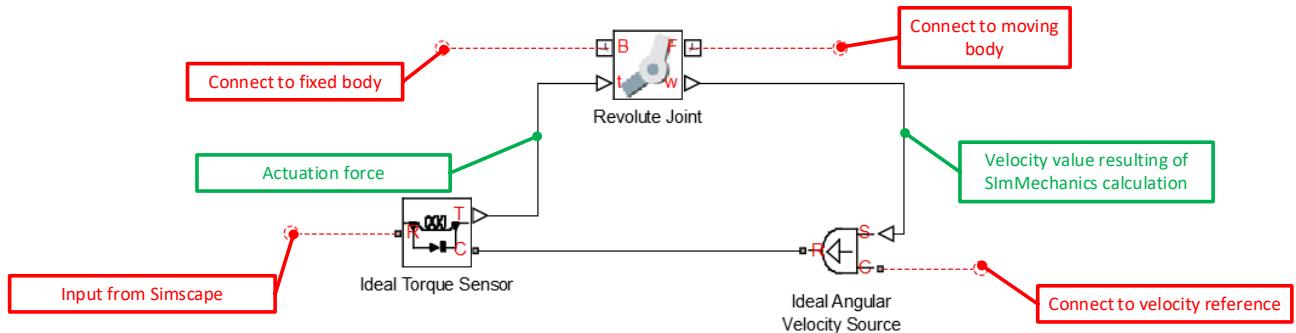


Figure 313: translational interface between Simscape and SimMechanics

2. Rotational interface between Simscape and SimMechanics

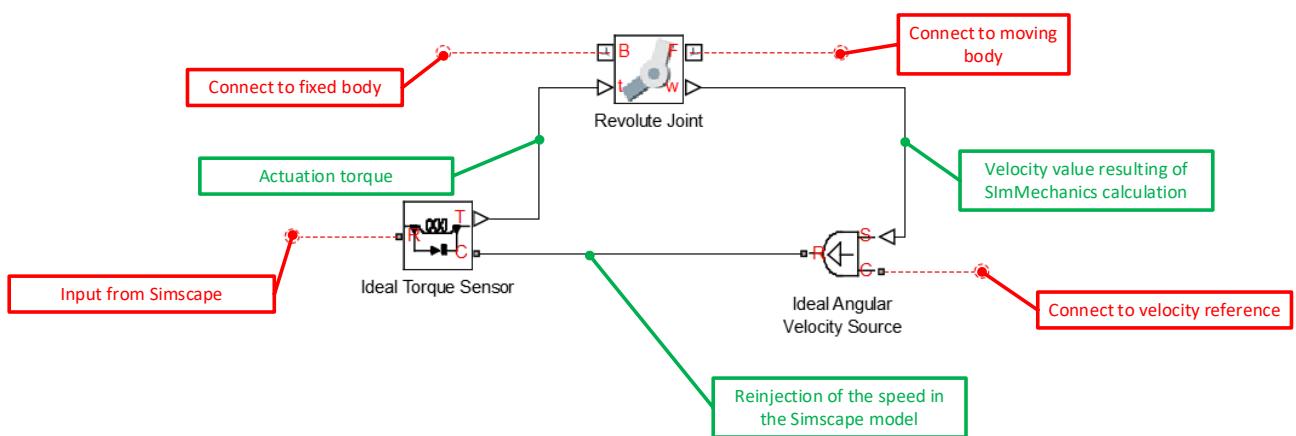
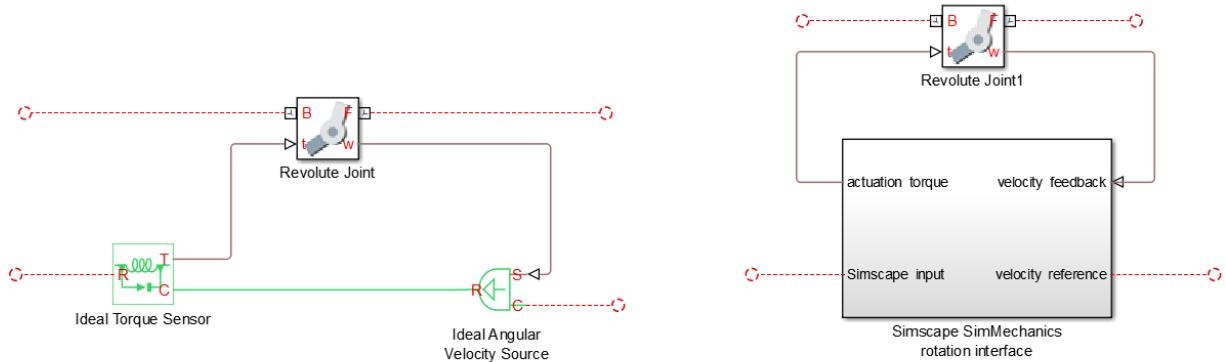


Figure 314: rotational interface between Simscape and SimMechanics

Open the file “**Simscape_SimMechanics_Interfaces.slx**”.

This file contains the interfaces between **Simscape** and **SimMechanics**. It is also possible to implement these interfaces in the form of sub-systems in order to easily re-use them.



Ivan LIEBGOTT@2016

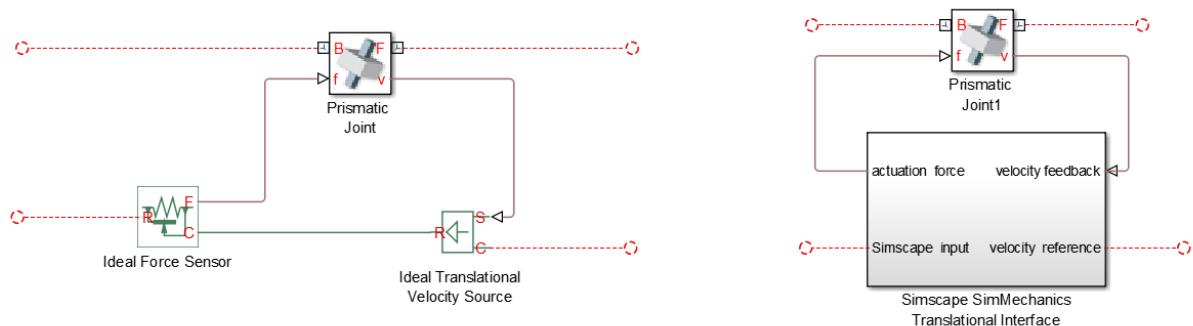


Figure 315: interfaces between Simscape and SimMechanics

Copy the Simscape SimMechanics Interface translational block into the model.

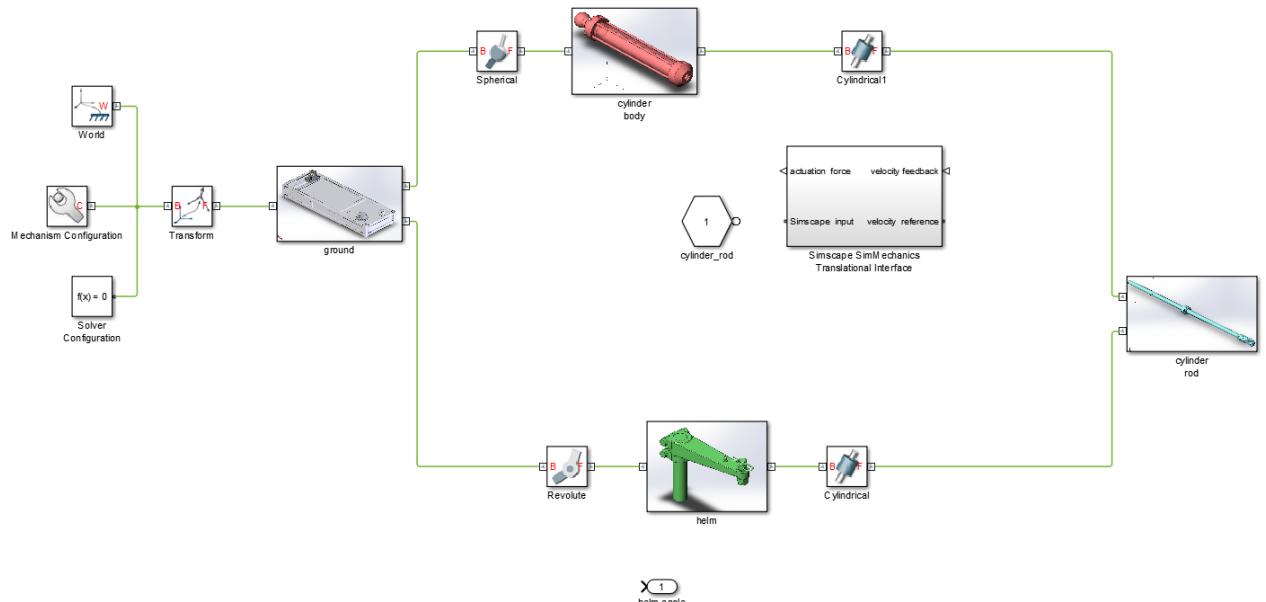


Figure 316: addition of Simscape SimMechanics Interface block

3. Adding ports to a connection

We now need to add connection ports to the connection block

- A port to enable it to be activated
- A port to return the speed value

Double-click on the cylindrical joint block to display the connection configuration dialogue box and configure the connection as in Figure 317 to create a port to activate the connection and a port to record the speed. Also set a viscous friction coefficient (Damping Coefficient) of 200 N.s/m.

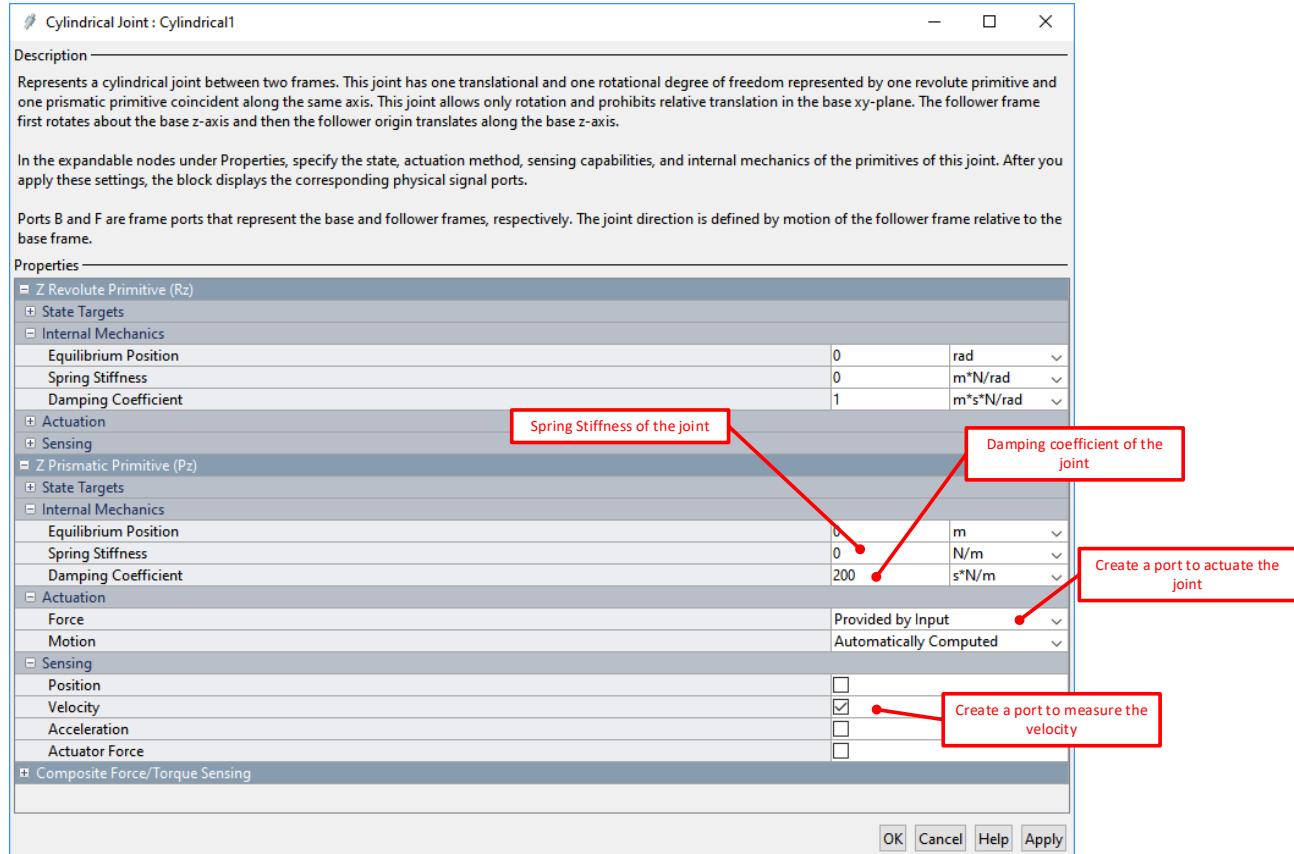


Figure 317: configuring the ports for a connection

Ports will appear on the connection block. Connect the interface as shown in **Erreur ! Source du renvoi introuvable..**

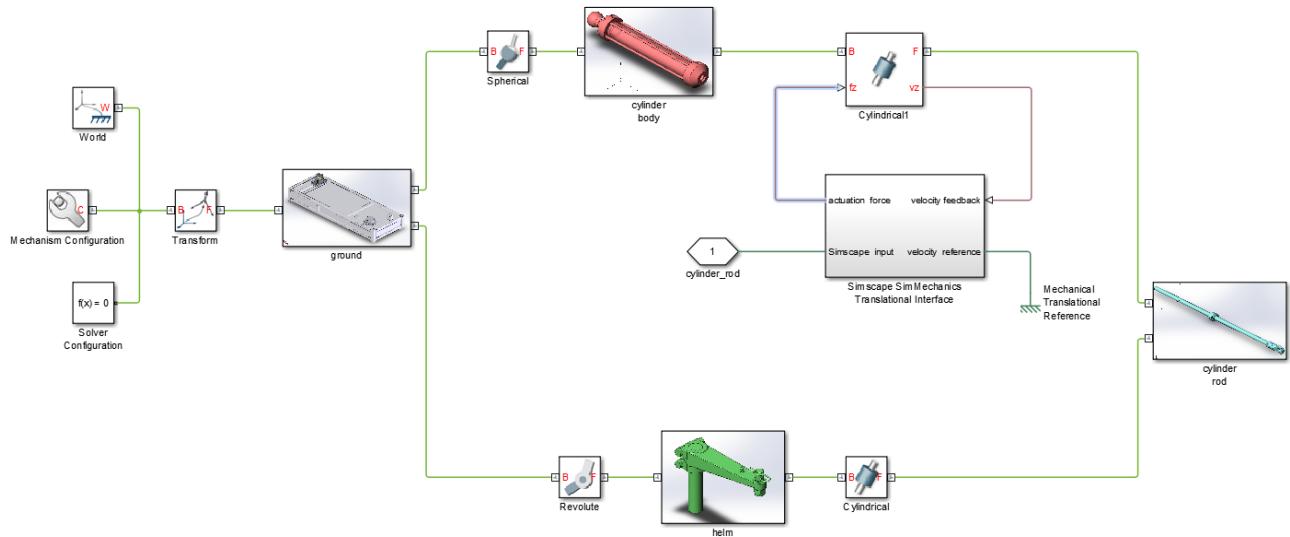


Figure 318: Connection between Simscape and SimMechanics

Since the interface between Simscape and SimMechanics has already been created, we are now going to work on the “Revolute” joint, between the “ground” and “helm”:

- A port to introduce a resistance force that acts on the helm of the boat.
- A port to read the rotation angle of the boat helm.

To do this, **double-click** on the connection “Revolute” and add a port to introduce a mechanical action on the connection as shown in Figure 319.

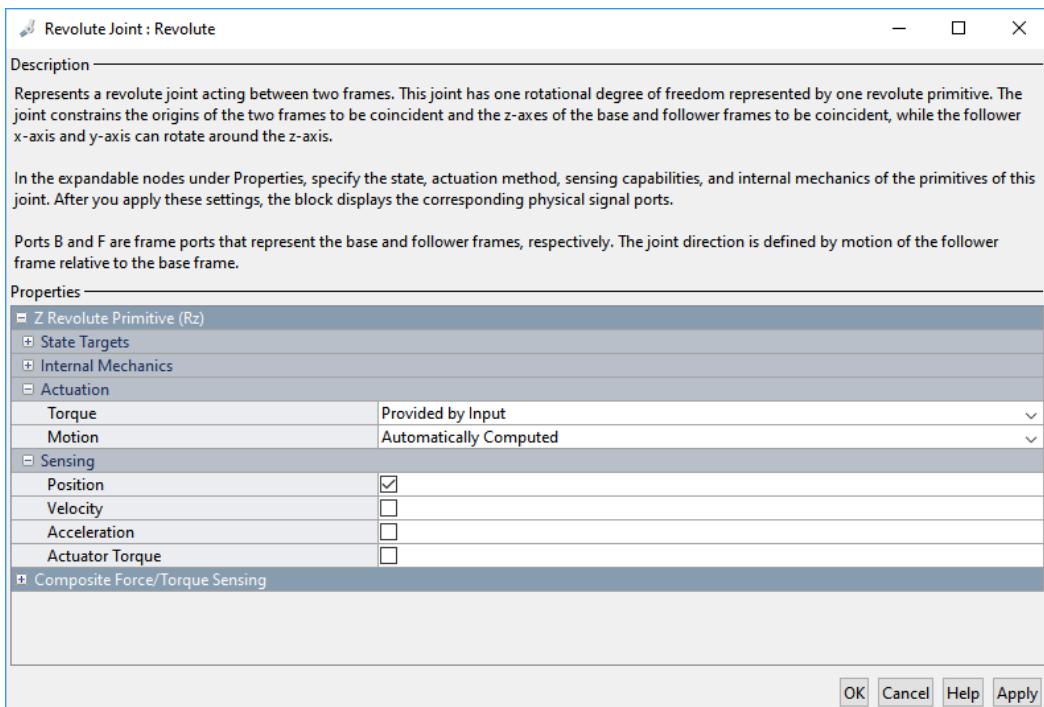


Figure 319: adding a port to introduce torque to a connection

The two new ports now appear on the connection. Add a **PS-S** block to convert the rotational angle signal of the helm into a **Simulink** signal. Choose degrees as the angle unit.

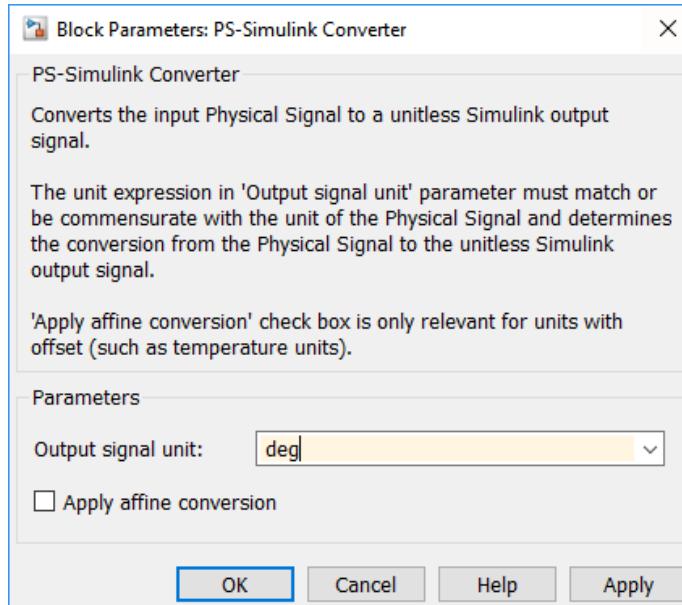


Figure 320: conversion of physical signal into a Simulink signal

You must obtain the configuration of Figure 321.

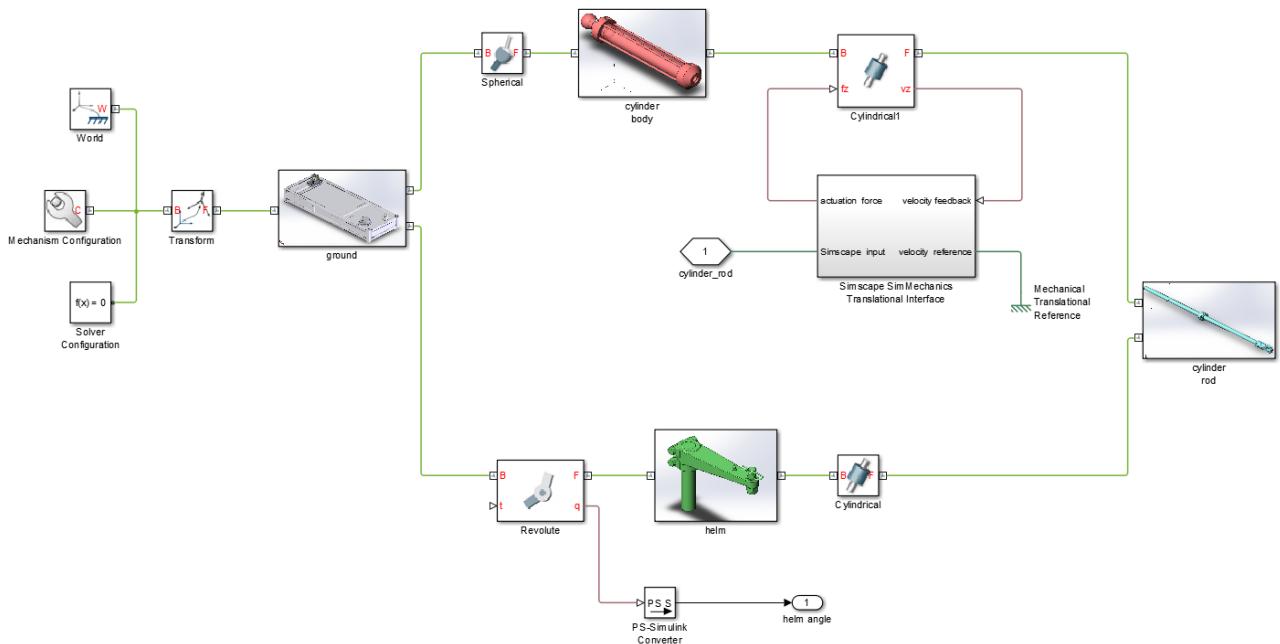


Figure 321: adding a port to a connection in SimMechanics

At the beginning of the simulation the angle of rotation of the helm has an arbitrary valued that is fixed when the model is imported from Solidworks. It is possible to find the initial offset value of the helm. To do this, double-click on the “Revolute” connection and expand **State Target/Specify Position Target**.

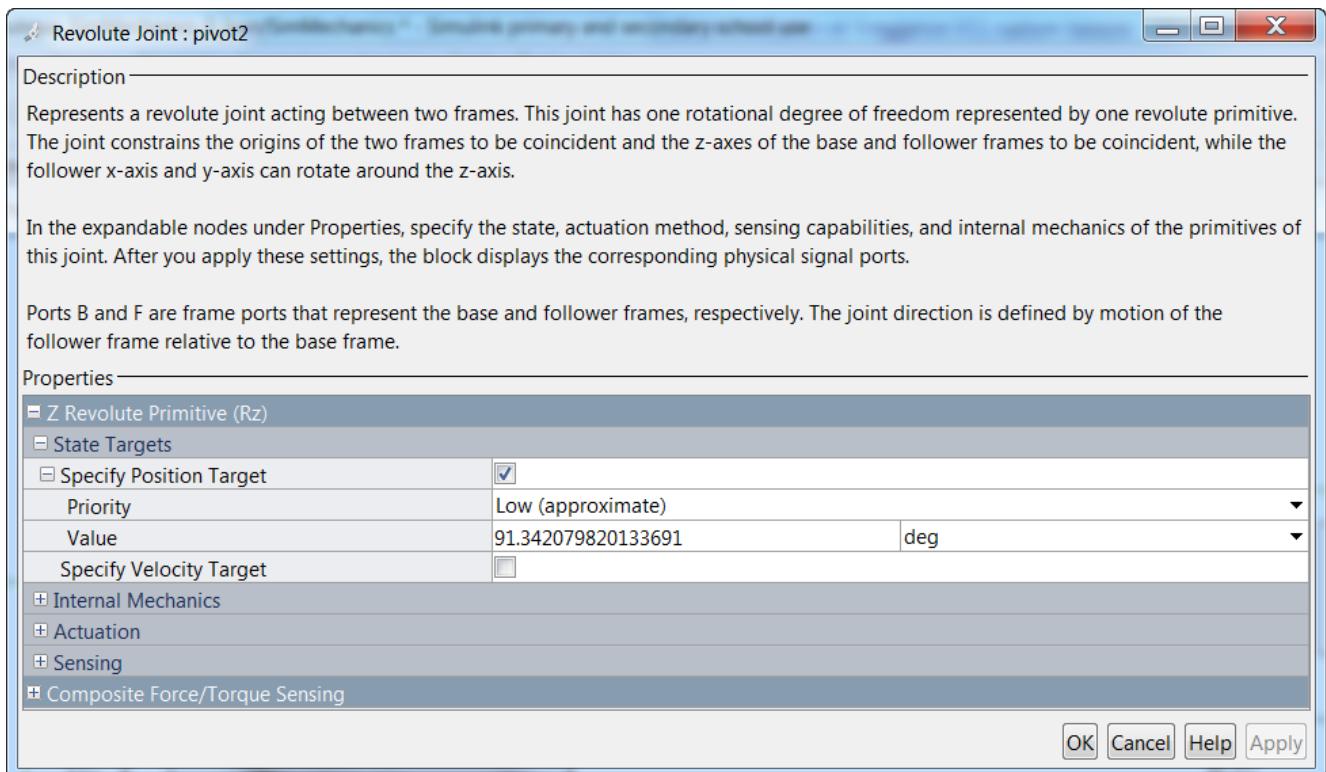


Figure 322: viewing the offset of the helm angle

At the beginning of the simulation, the helm angle should be zero and so this offset should be subtracted from the angle value returned by the connection. To do this, **complete** the model as shown in Figure 323 by adding a **constant** block that you can fill in by copying and pasting the offset value of the helm angle and a subtractor to subtract the offset from the signal.

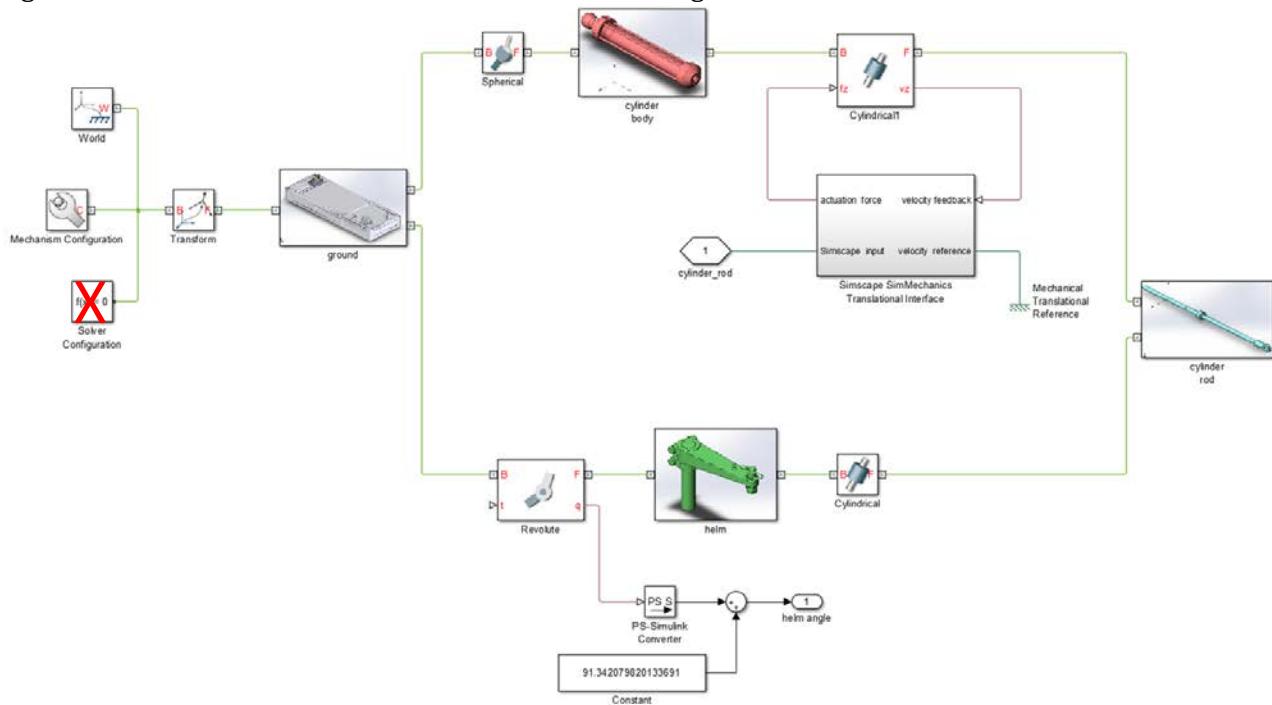
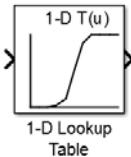


Figure 323: compensating for the offset of the helm angle

You must delete the solver configuration block, because only one of this block could be needed in the model.

4. Modeling a variable external force

The time that opposes the rotation of the helm during the movement varies according to the rotation angle of the helm. Since this variation is non-linear, we will use a **1D Lookup Table** to model it. The **1D Lookup Table** represents a gain that may vary depending on the input value. This block is very useful for modeling non-linear geometric or dynamic input-output laws.

Designation	View	Library
1D Lookup Table		Simulink/ Lookup Tables

Open the file **hydraulic_pilot_SimMechanics_end_US.slx** and see the addition of the non-linear force to the connection. The helm angle is read and the model introduces a force on "pivot2" that varies according to the helm angle.

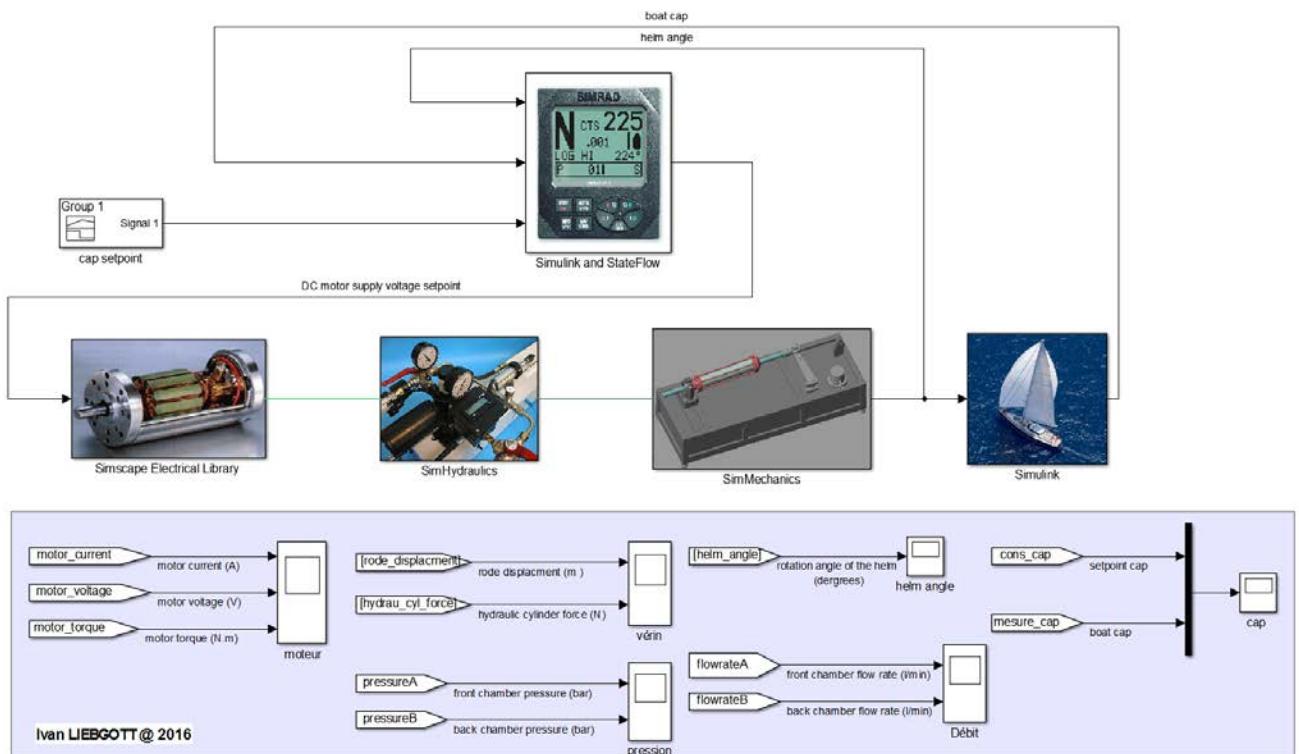


Figure 324 : *hydraulic_pilot_SimMechanics_end_US.slx*

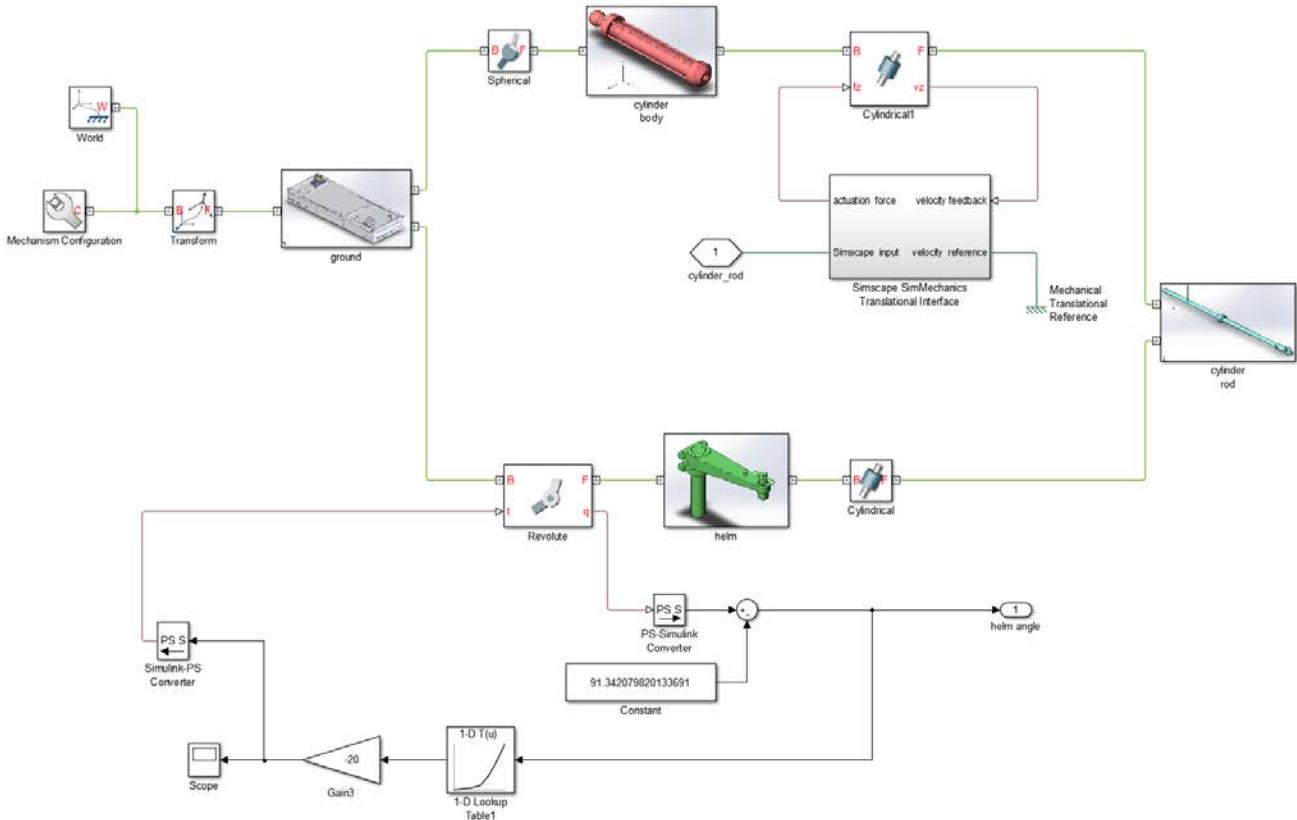


Figure 325: adding a force to a connection, using a Look-up Table

Double-click on the **1D Look-up Table** block to explore the configuration of this block type.

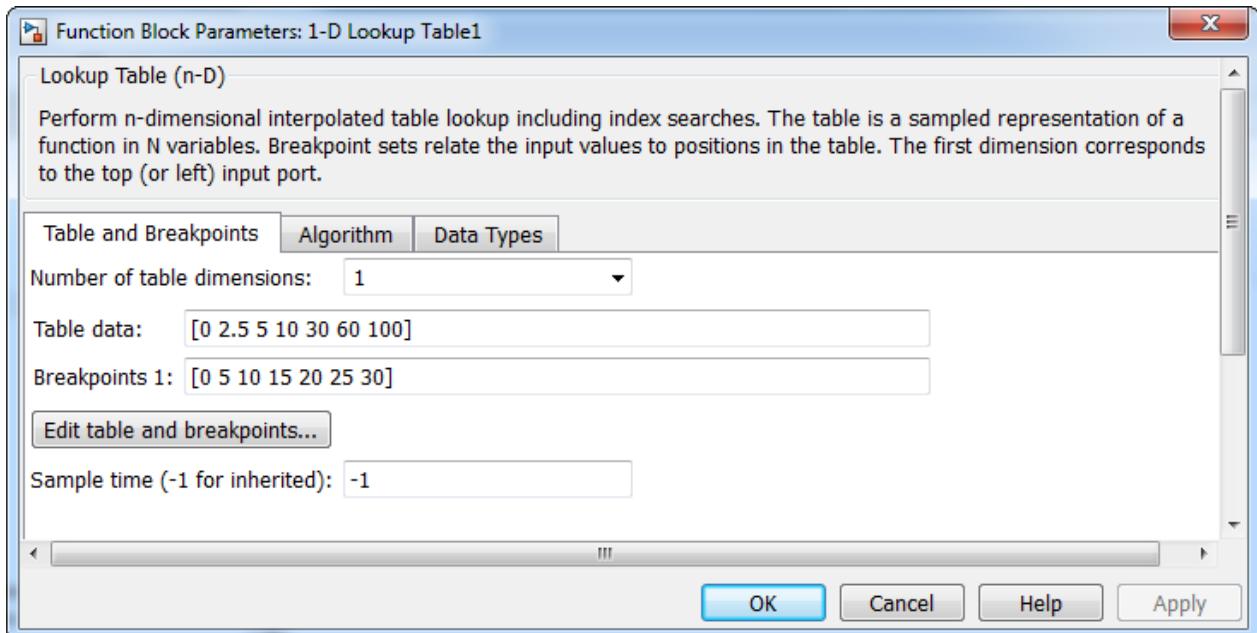


Figure 326: configuring a Look-up Table

To configure the table, simply enter the points belonging to the curve which defines the input-output law of the block. The software automatically interpolates this to combine the corresponding output value with the associated input value.

The point coordinates are entered into the **Breakpoints 1** (abscissa) and **Table data** (ordinate) fields.

It is possible to visualize the law entered by clicking on **Edit table breakpoints**.

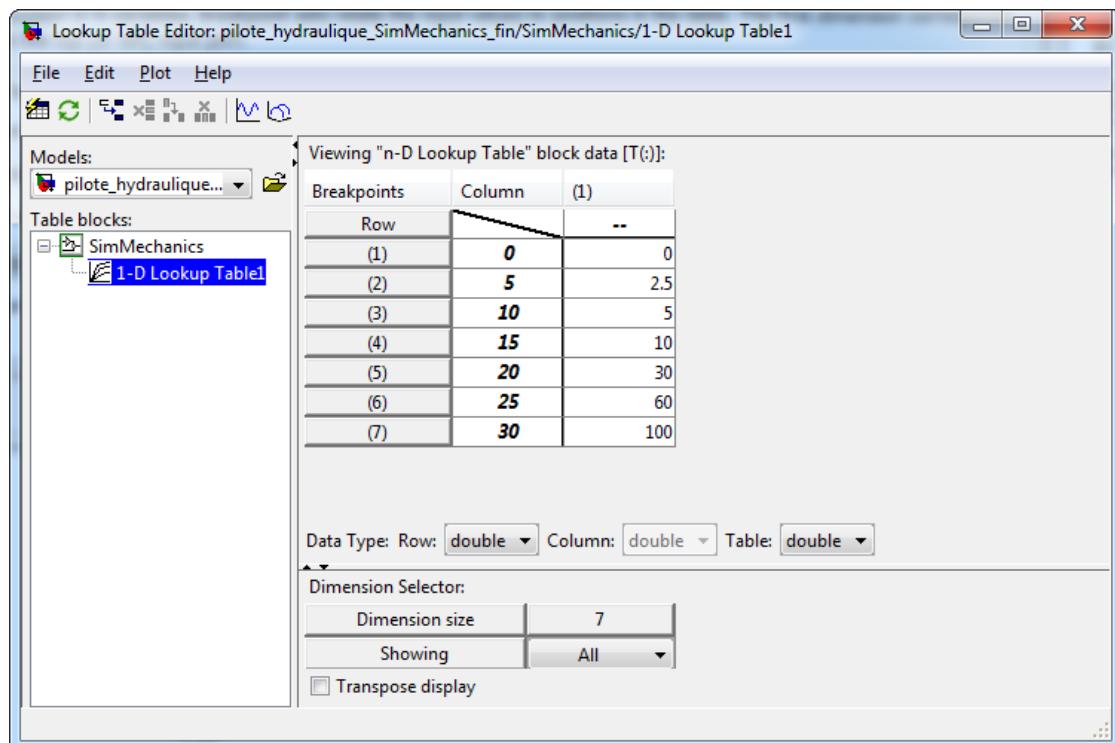


Figure 327: viewing a Lookup Table

It is possible to visualize the curve representing the input-output law by clicking on **Linear Plot** .

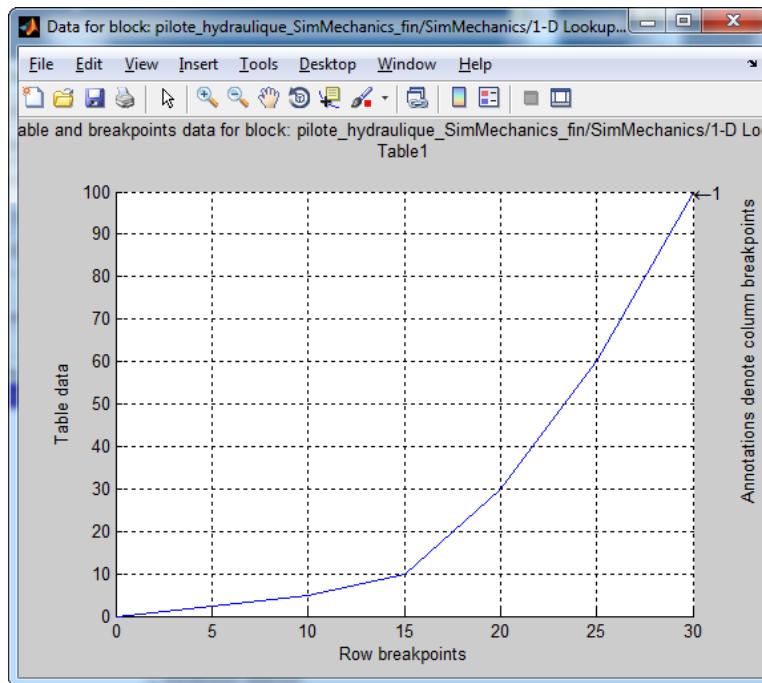


Figure 328: view of the curve representing the input-output law for a Lookup Table

Validate the configuration of the Lookup Table.

C. Results of the simulation

Launch the simulation and observe the response in the boat heading which takes into account the added **SimMechanics** model.

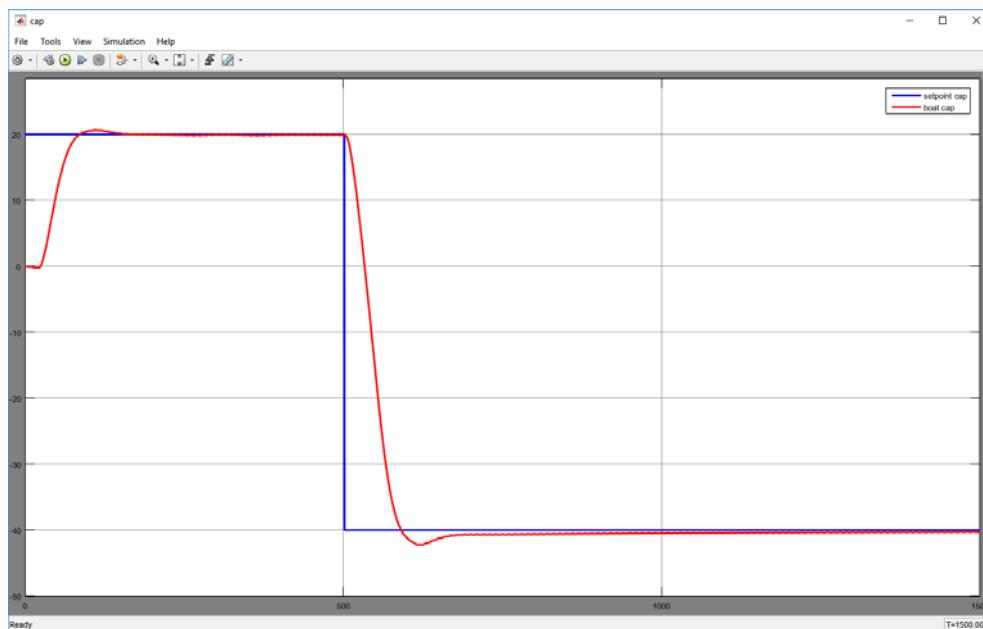


Figure 329: Results of the simulation of the course followed by the boat

III. Importing a SolidWorks model into SimMechanics

A. Principles

To operate in **MATLAB** – Simulink CAD 3D model from Solidworks, convert this model into a file with an **xml** extension. To do this, you must first install a “**SimMechanics Link**” add-on in SolidWorks to complete the conversion of a Solidworks assembly file into an xml file. The xml file is then converted by **MATLAB** into a usable file for SimMechanics.

During conversion each Solidworks file “part” will be converted to an STL file. These files contain the shapes of parts and kinetic characteristics. The presence of these files in the **MATLAB** “path” is required to view animations in the **Mechanics Explorer** window.

B. Installing “SimMechanics Link”

To download the **SimMechanics Link**, you must have a MathWorks account. You will need to create a MathWorks account if you do not already have one.

Solidworks and MATLAB must be installed on your computer.

Sign in with your login and password.

Go to the following address:

http://www.mathworks.com/products/simmechanics/download_smlink.html

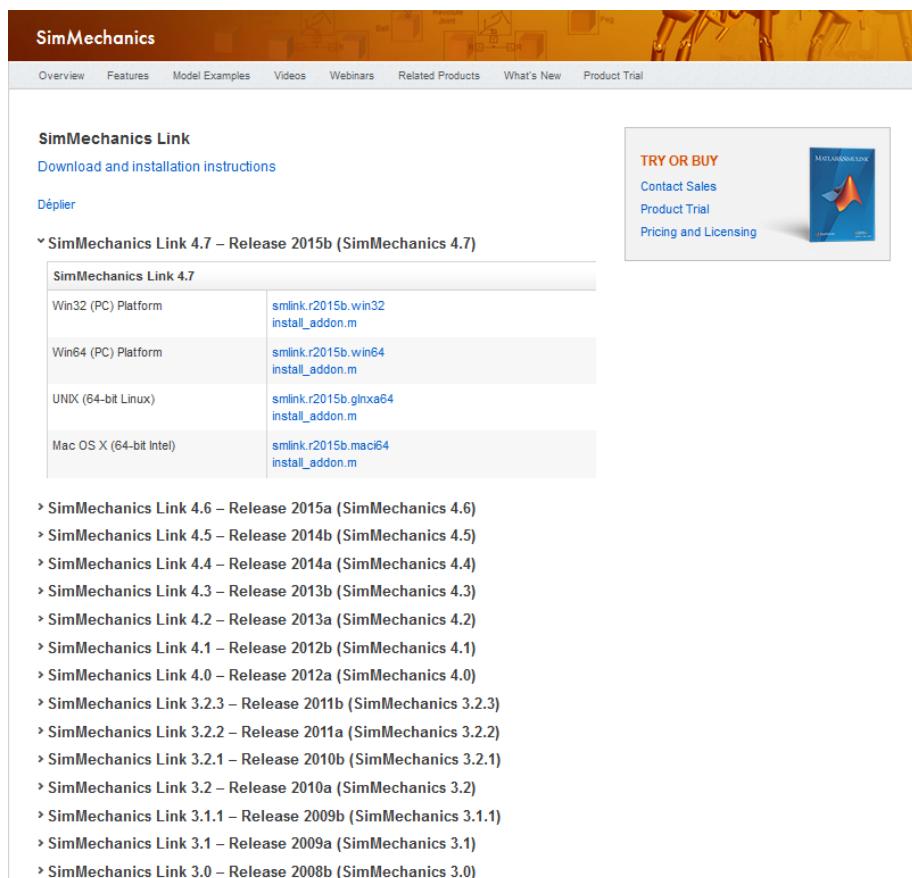


Figure 330: Choosing the SimMechanics Link version

You must choose the version of **SimMechanics Link** that corresponds with your version of **MATLAB** and download the two files for your operating system, ensuring you save them in a folder on the **MATLAB** “path”.

- MATLAB script file: **install_addon.m**
- File archive containing files to install: **smlink.r...**

In the MATLAB command window type the command which will launch the installation of SimMechanics Link:

```
>> installAddon('smlink.r2015b.win64.zip')  
.....
```

Installation of smlink complete.

To view documentation, type "doc smlink"

Please note, the name of the file archive in the parentheses of the **installAddon** command must exactly match what you downloaded and will depend on your version of MATLAB and your operating system.

Once installation has completed, type the following command to create the link between MATLAB and Solidworks.

```
>> smlink_linksw
```

The installation process is complete.

C. Converting a Solidworks assembly file to xml file

Launch Solidworks.

The software welcome window opens

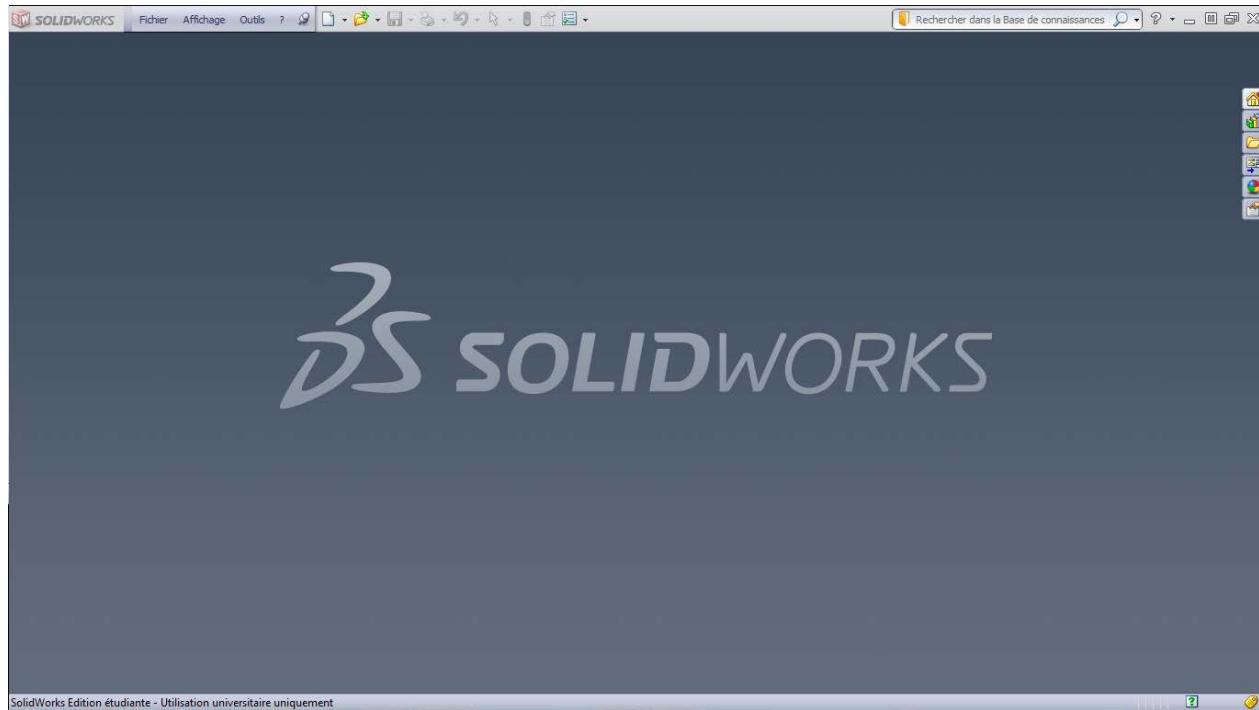


Figure 331: Solidworks welcome window

Launch Solidworks.

The software home window opens, open the drop-down **options** menu then select add-ons to open the software add-on definition window.

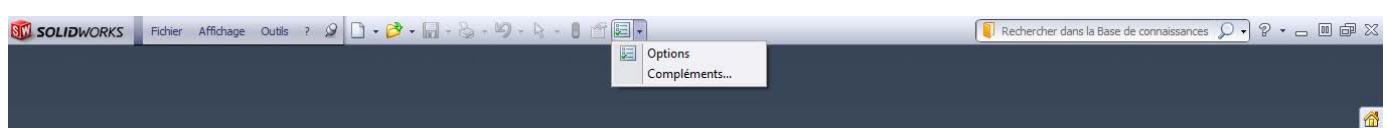


Figure 332: Opening the Solidworks add-on window

Check both left and right boxes of **SimMechanics Link** to activate **SimMechanics Link** each time you start Solidworks (Figure 333).

Click **OK**.

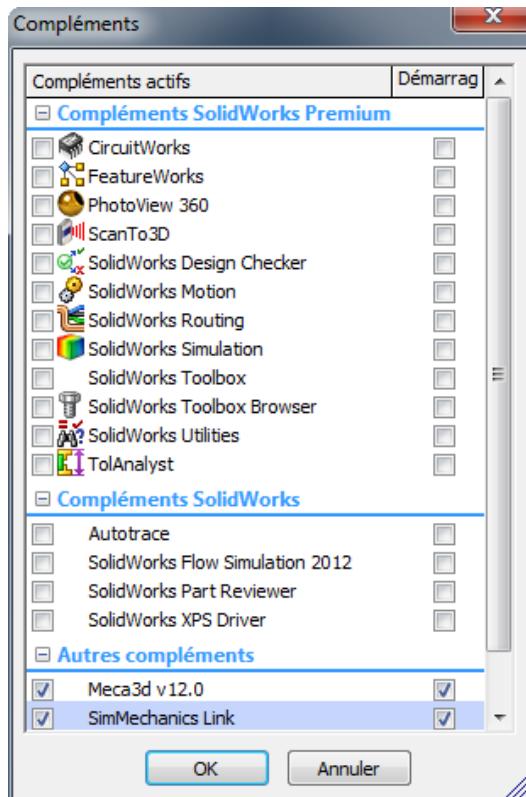


Figure 333: Selecting Solidworks add-ons

Exit and then **restart** Solidworks.

With Solidworks, **Open** the *Assemblage pilote hydraulique.sldasm* file in the “CAD 3D Model Driver/Mechanical Part” folder

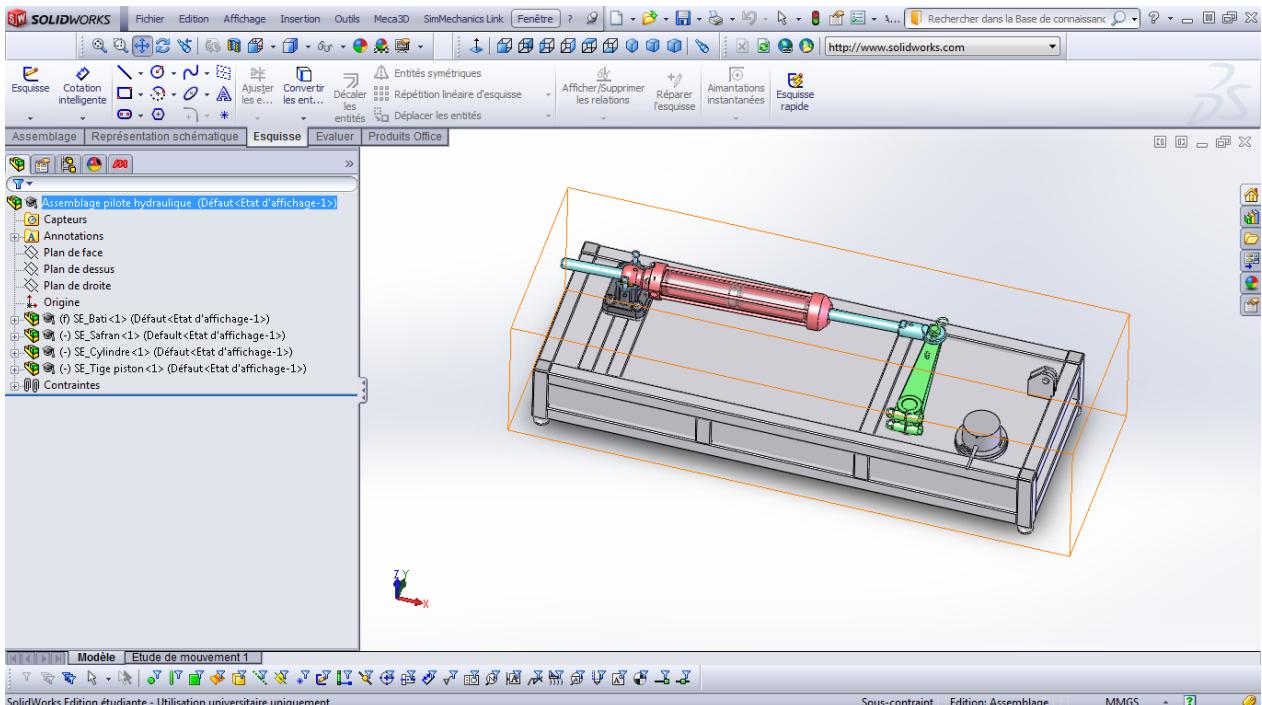


Figure 334: Opening the Solidworks model of the mechanical part of the pilot

To convert the assembly to **xml** format, select **SimMechanics Link/export/SimMechanics Second Generation** (Figure 335). Make sure to choose a file that belongs to the MATLAB “path” as the export folder. Here we take the empty folder “conversion_pilote-xml” located at the same level as the “CAD 3D Model Driver” folder. Name the folder ***Assemblage_pilote_hydraulique***.

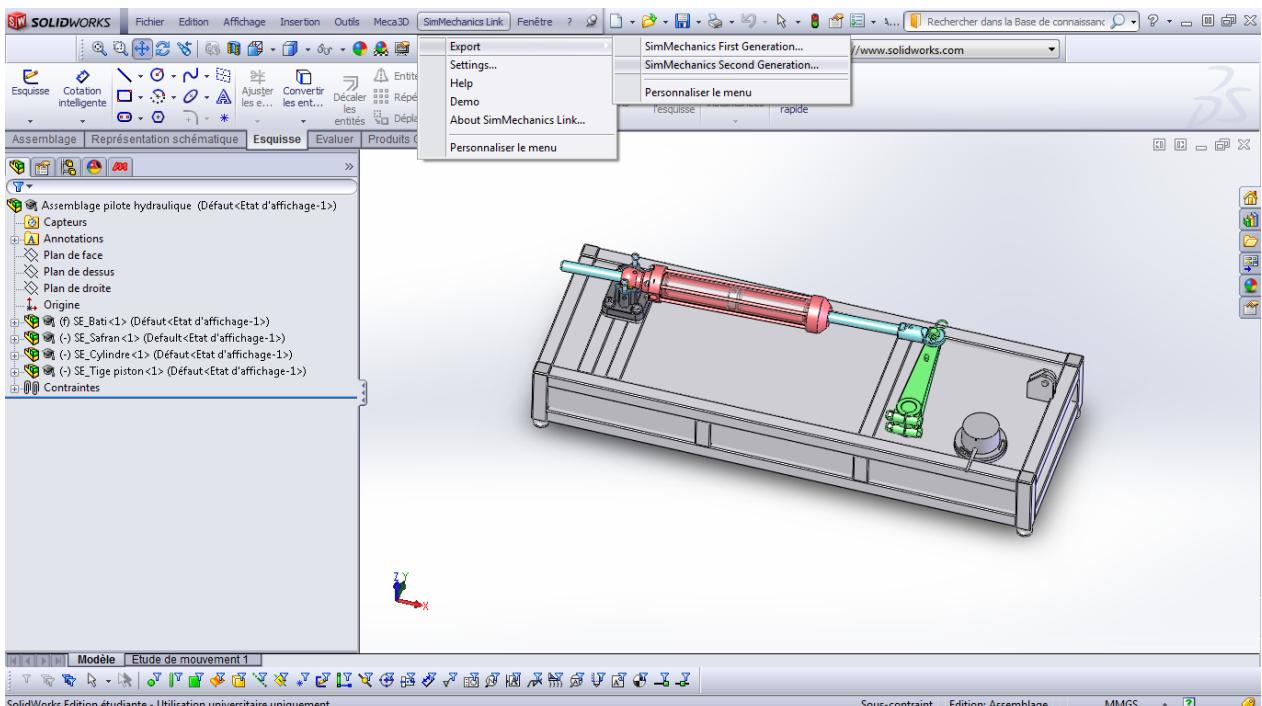


Figure 335: Converting the assembly to an xml file

After several seconds, Solidworks will open and close all the file parts that constitute the assembly to convert them to STL format. The window in Figure 336 indicates the end of the conversion.

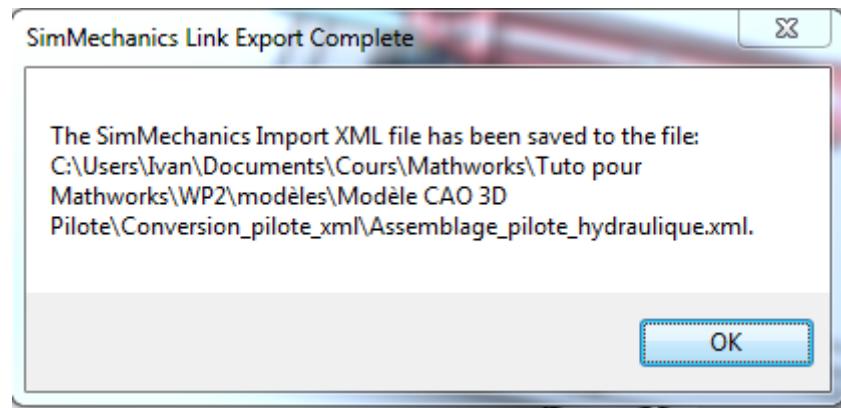


Figure 336: End of the file conversion

Open the “conversion_pilote_xml” folder with Windows Explorer to check that it contains all the parts of the mechanism in **STL** format as well as a file named **Assemblage_pilote_hydraulique.xml**.

Return to MATLAB and type into the command window:

```
>>smimport('Assemblage_Pilote_hydraulique.xml')
```

This command will allow the creation of a **SimMechanics** file corresponding to the assembly and automatically open a new file containing the model.

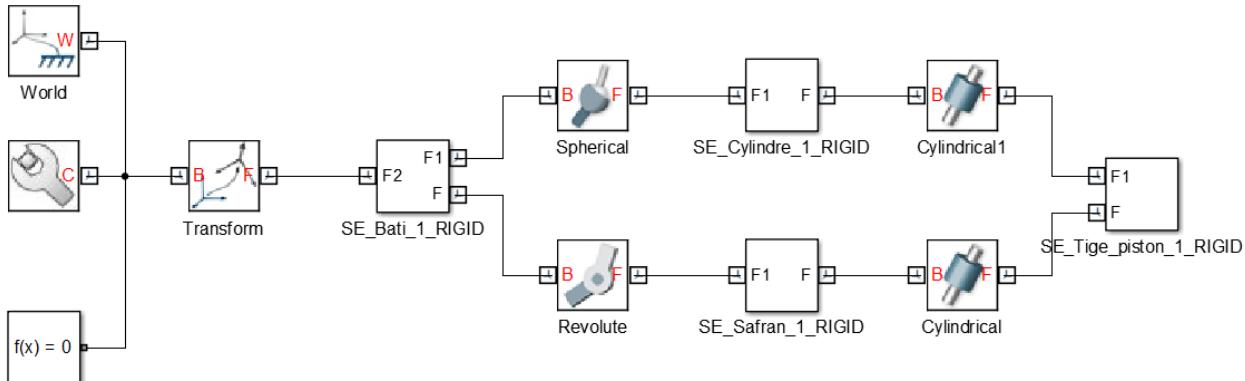


Figure 337: SimMechanics model obtained

We find the file on which we worked at the beginning of this chapter.

Launch the simulation, the Mechanics Explorer window will open and you can view the pilot model in MATLAB. The gravity may not be on the right axis by default.

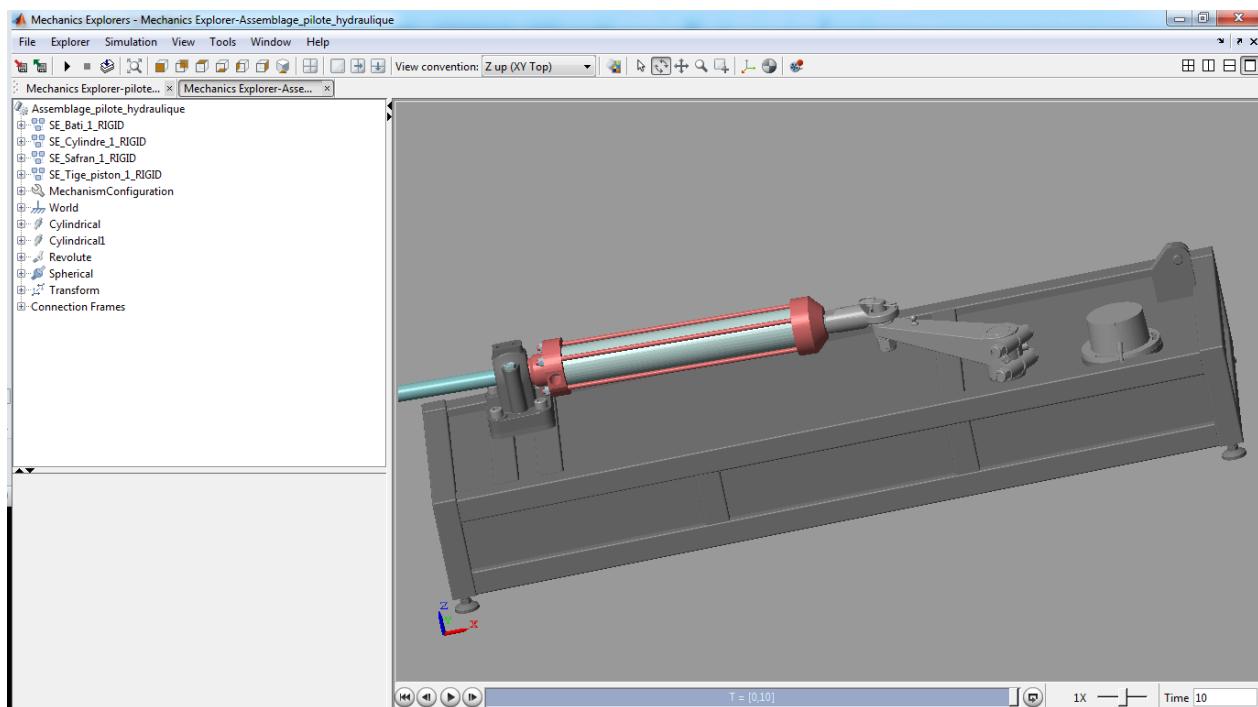


Figure 338: Mechanics Explorer window

Chapter 8: Identifying a model

I. Black-box modeling, identification

A. The model

This approach consists of associating a mathematical model with a measured experimental behavior. Here the system is considered as a black box with an input variable and an output variable. The idea is to input a known input variable to the system and to raise the output variable. We then need to find a mathematical model for the system transfer function that will yield the same relationship between input and output. This method provides only a model of the global behavior of the system, without delving into the separate influences of the various performance parameters.

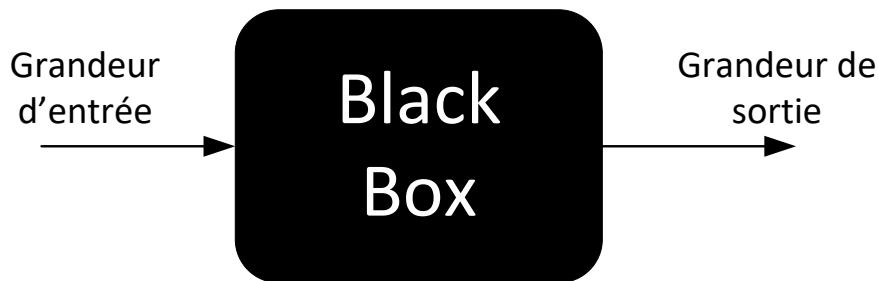


Figure 339: “Black box” modeling

This method provides a model experimentally validated without having to write the equations that govern the behavior of the system. The model should be treated with caution in case non-linear phenomena enter into the actual system behavior. In practice we always perform several tests corresponding to different constraints (constant or sinusoidal input value with a frequency variation).

Aside from simple cases, identification of the transfer function requires significant computational resources. The software tool will enable us to automate this task by suggesting mathematical models for identification processes that could be complex.

MATLAB provides a toolbox called “System Identification” which allows this process to be carried out.

B. Implementing the method using the Identification toolbox

1. Analysis of the data used for identification

Open and run the script **identification_data_US.m**.

This script displays the input and output variables of a test in an open-loop on a linear axis. A voltage is imposed on the motor terminal of the linear axis (input variable) and the linear velocity of the axis is raised (output variable). The conditions under which the test was performed are illustrated in Figure 340.

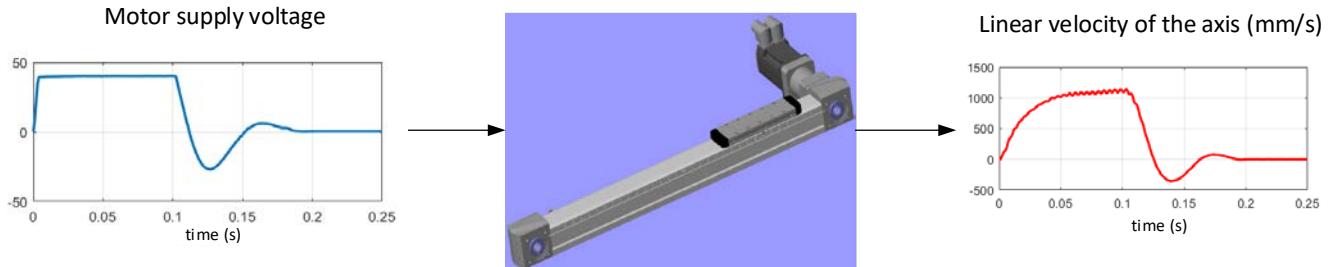


Figure 340: Test conditions

Figure 341 displays the input and output signals.

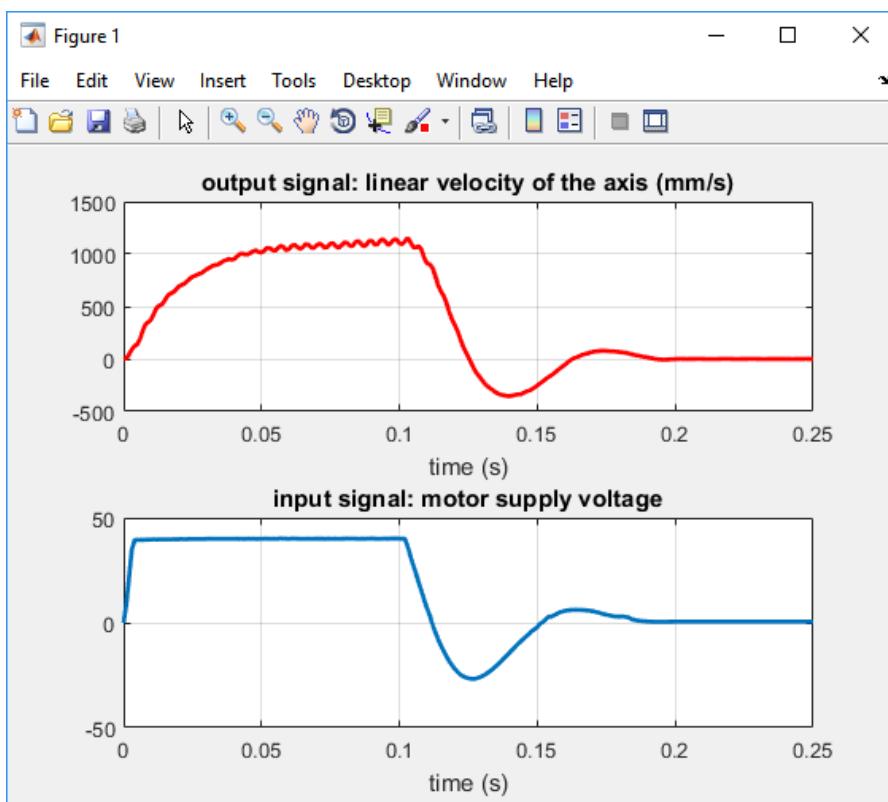


Figure 341: Display of the data used for identification

Executing the script in MATLAB **Workspace** also creates the following variables:

Name	Value
Step_response	251x17 double
t	251x1 double
u	251x1 double
v	251x1 double
x	251x1 double

Figure 342: Display of the data used for identification in Workspace

- t: vector containing variable time
- u: vector containing the variable voltage supply
- v: vector containing the variable linear velocity of the axis
- x: vector containing the variable linear position of the axis

2. Opening and presentation of the “System Identification” toolbox

To run the “System Identification” toolbox, type **ident** in the command window.

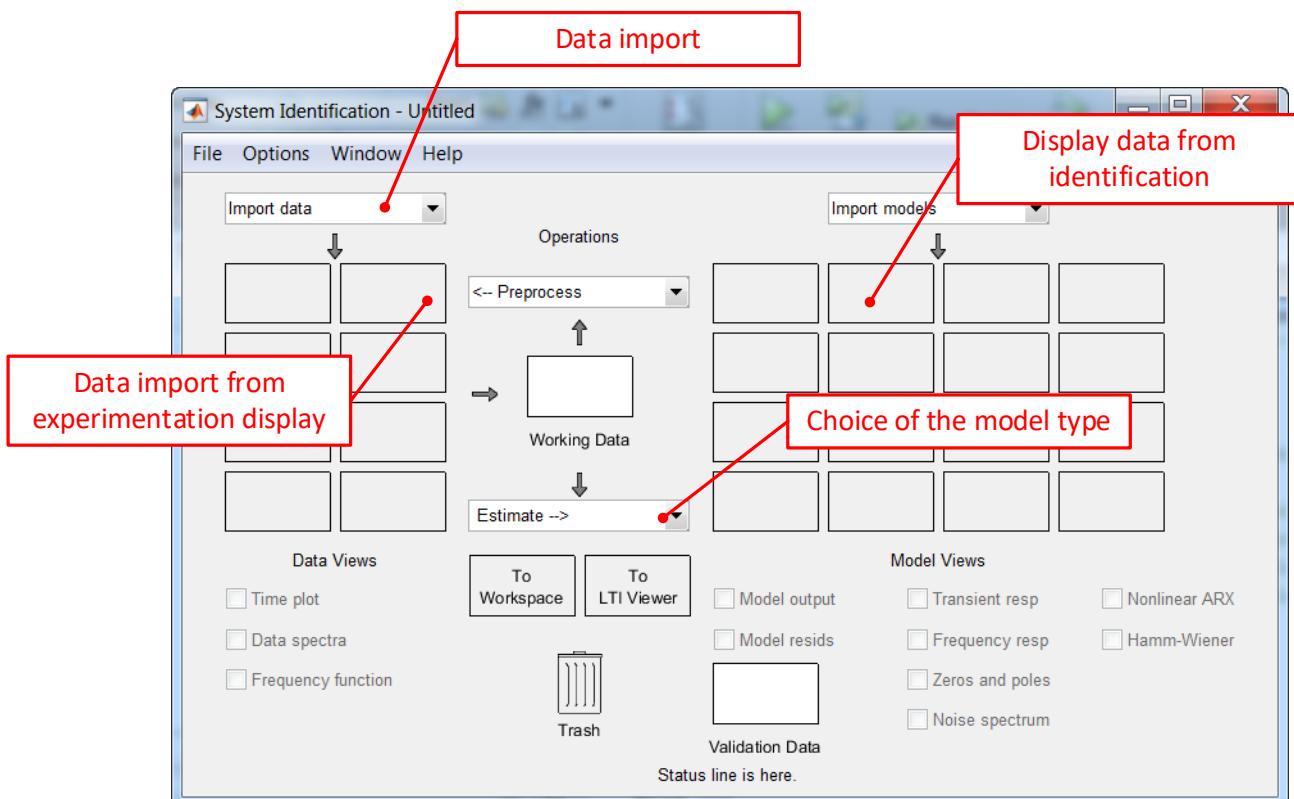


Figure 343: Description of the toolbox identification window

3. Importing data

It is possible to import any type of data: time, frequency, object data...
In our example the data from the experiment is time data.

Select **Import data/Time domain data**.

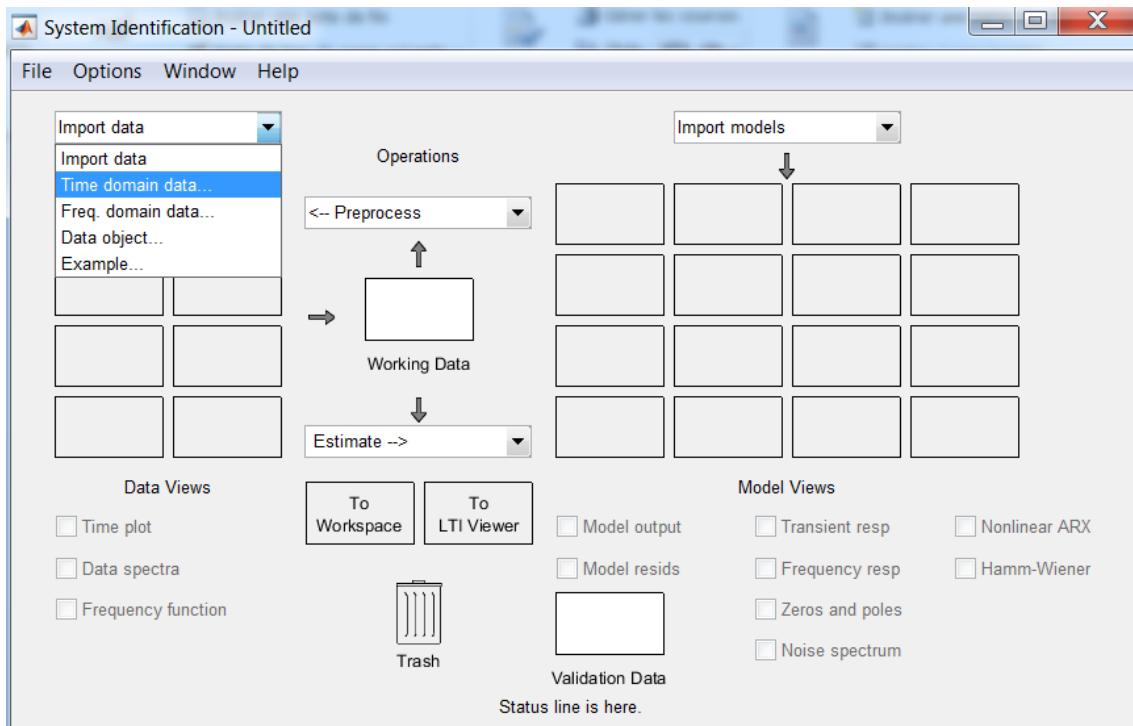


Figure 344: Importing time domain data

This command opens the window to define the data necessary for identification. Complete this window with the data from Figure 345.

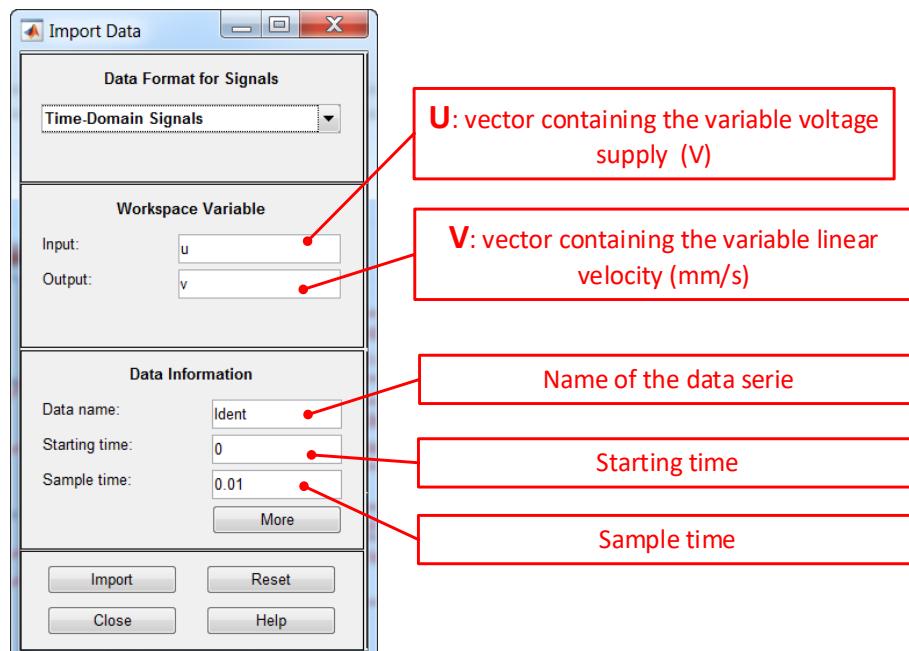


Figure 345: Selection of time domain data for identification

The data will appear in the “System Identification” toolbox window as shown in Figure 346. Check the **Time plot** box to verify and display the proper importation of the data (Figure 347).

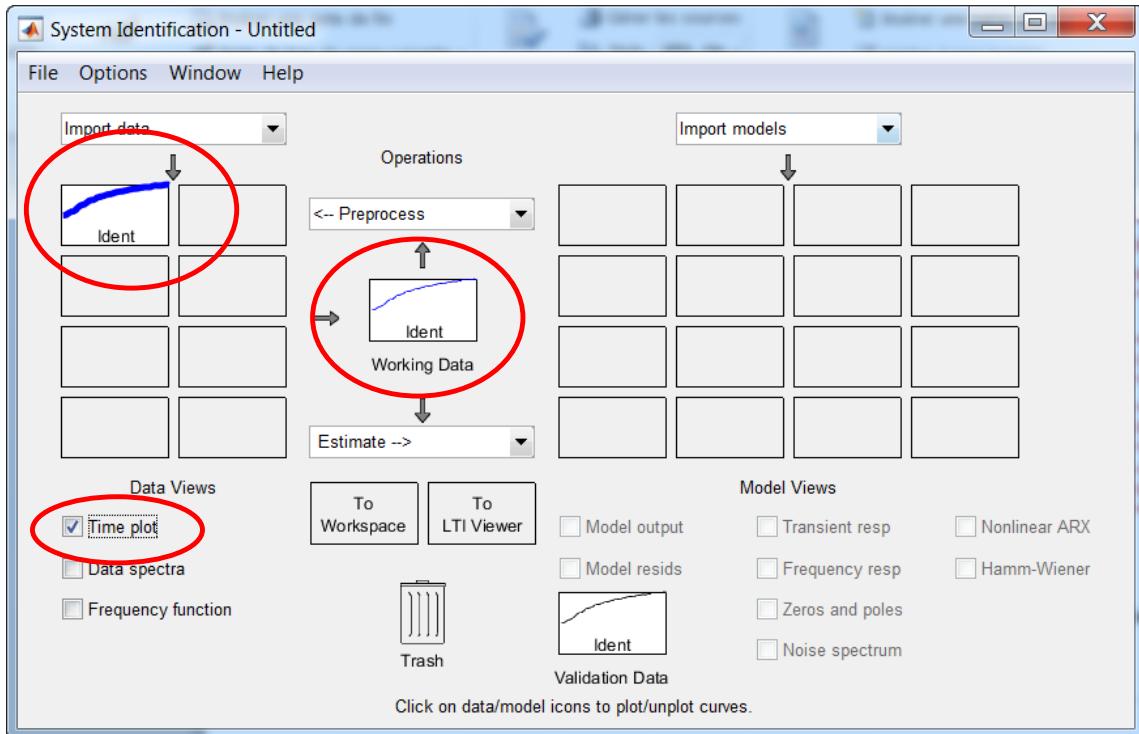


Figure 346: Display of imported data in the “System Identification” toolbox window

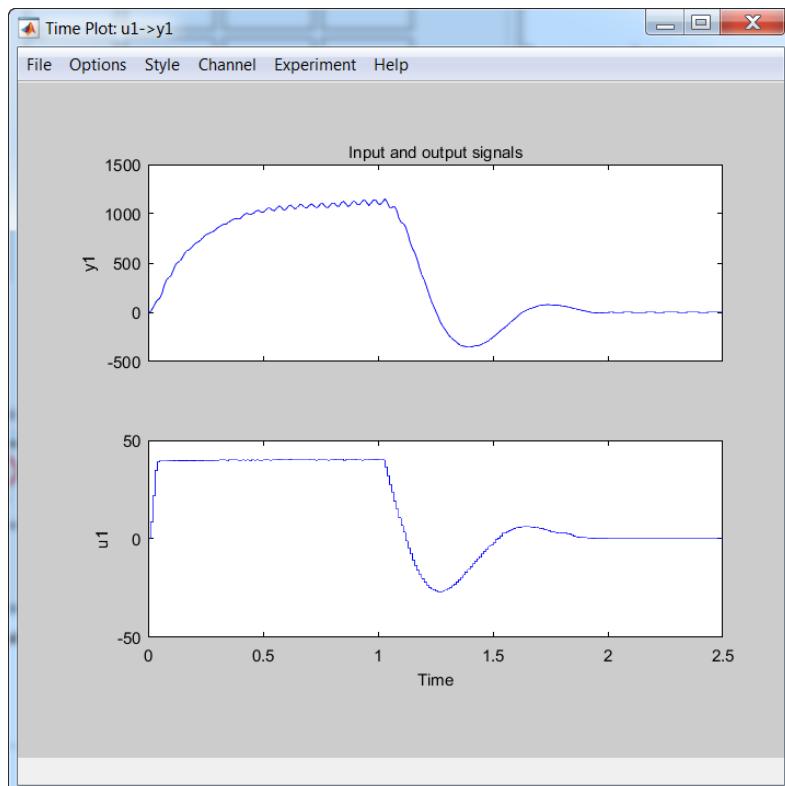


Figure 347: Display of imported data in the “System Identification” toolbox

We must now choose the desired model type (transfer function, state-space, non-linear model...). Here we must choose a **transfer function** type model.

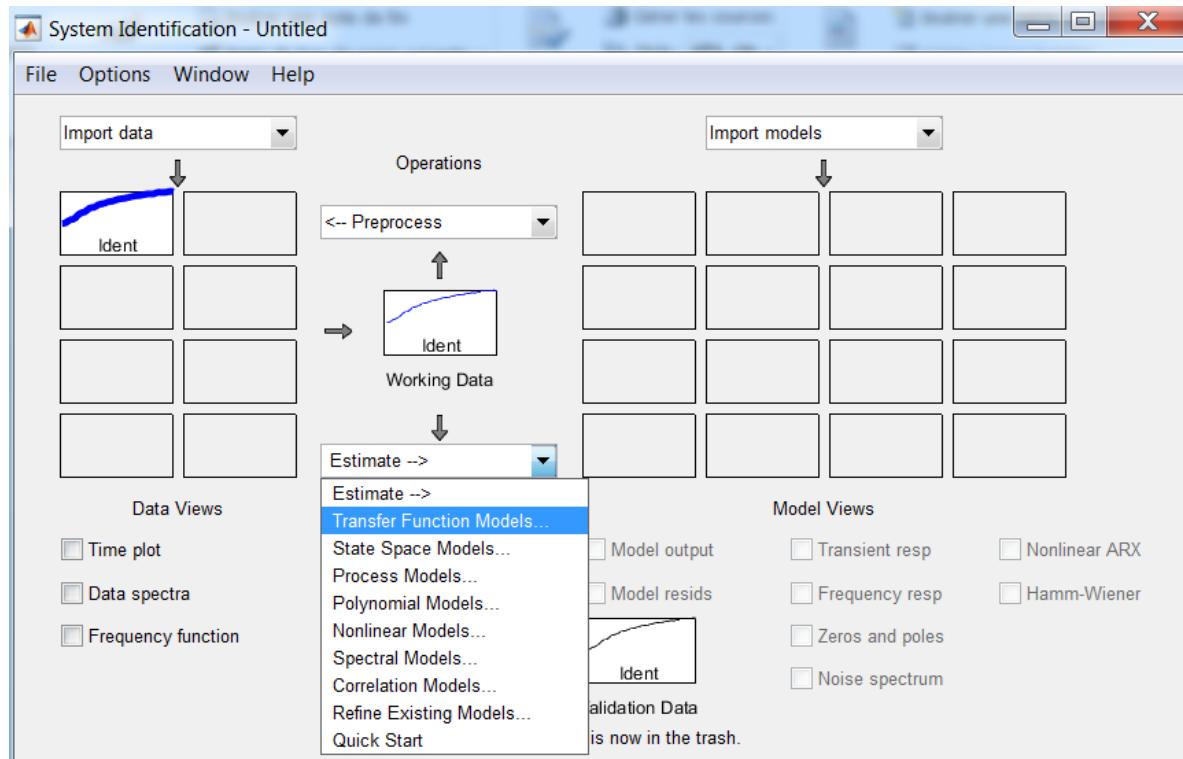


Figure 348: Choosing the model type in the “System Identification” toolbox

The window that opens allows you to choose the desired type of transfer function. Here we will use a second order transfer function. Select **2 poles and no zero** and click **Estimate**.

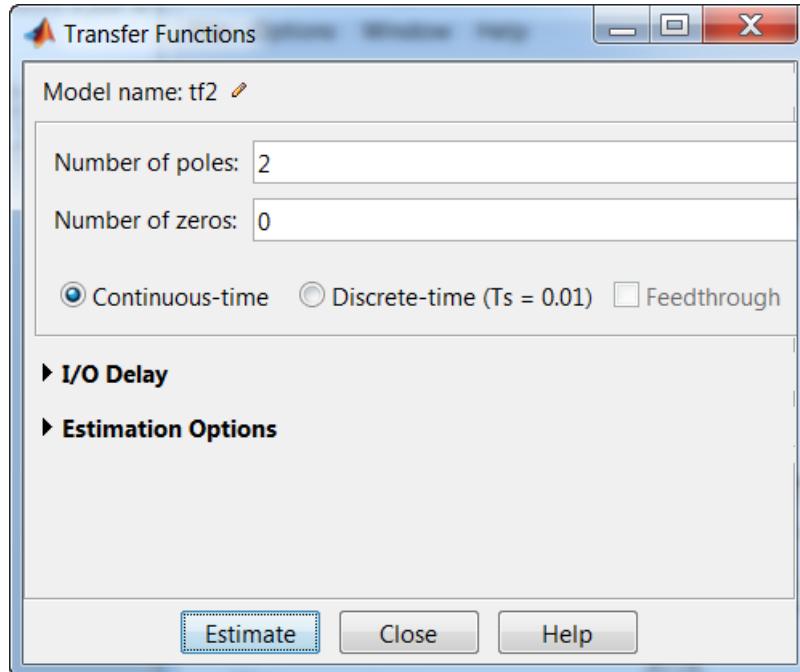


Figure 349: Choosing the transfer function form in the “System Identification” toolbox

The **Plant Identification Progress** window gives the details of the numeric identification process

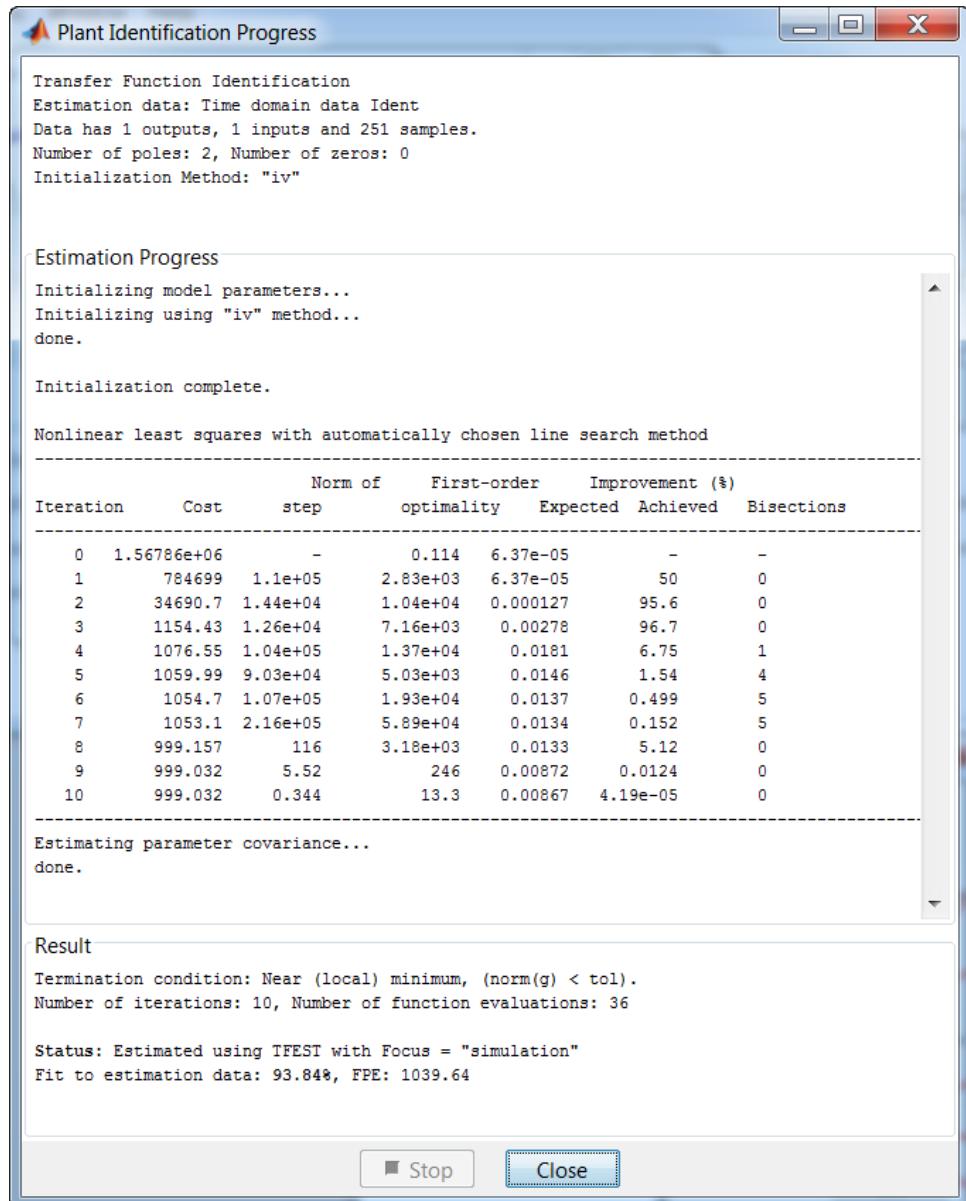


Figure 350: The Plant Identification Progress window of the “System Identification” toolbox

The result of the identification is available by right-clicking the mouse on the **tf1** box representing the identification results (Figure 351 and Figure 352). Also check the **Model output** box to view the overlap of the experimental data and the model resulting from the identification (Figure 353).

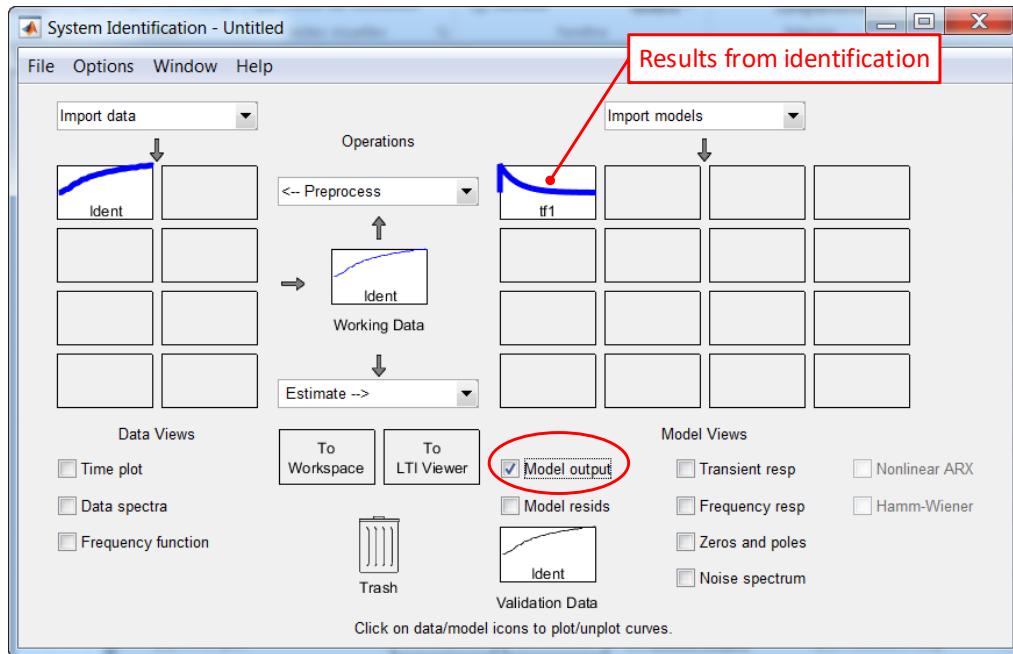


Figure 351: Displaying the identification results

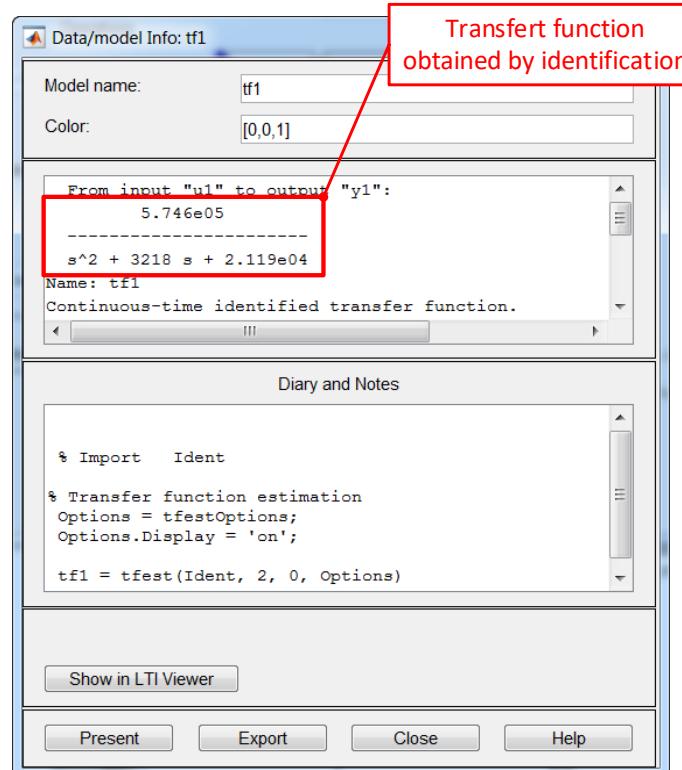


Figure 352: Transfer function obtained by identification

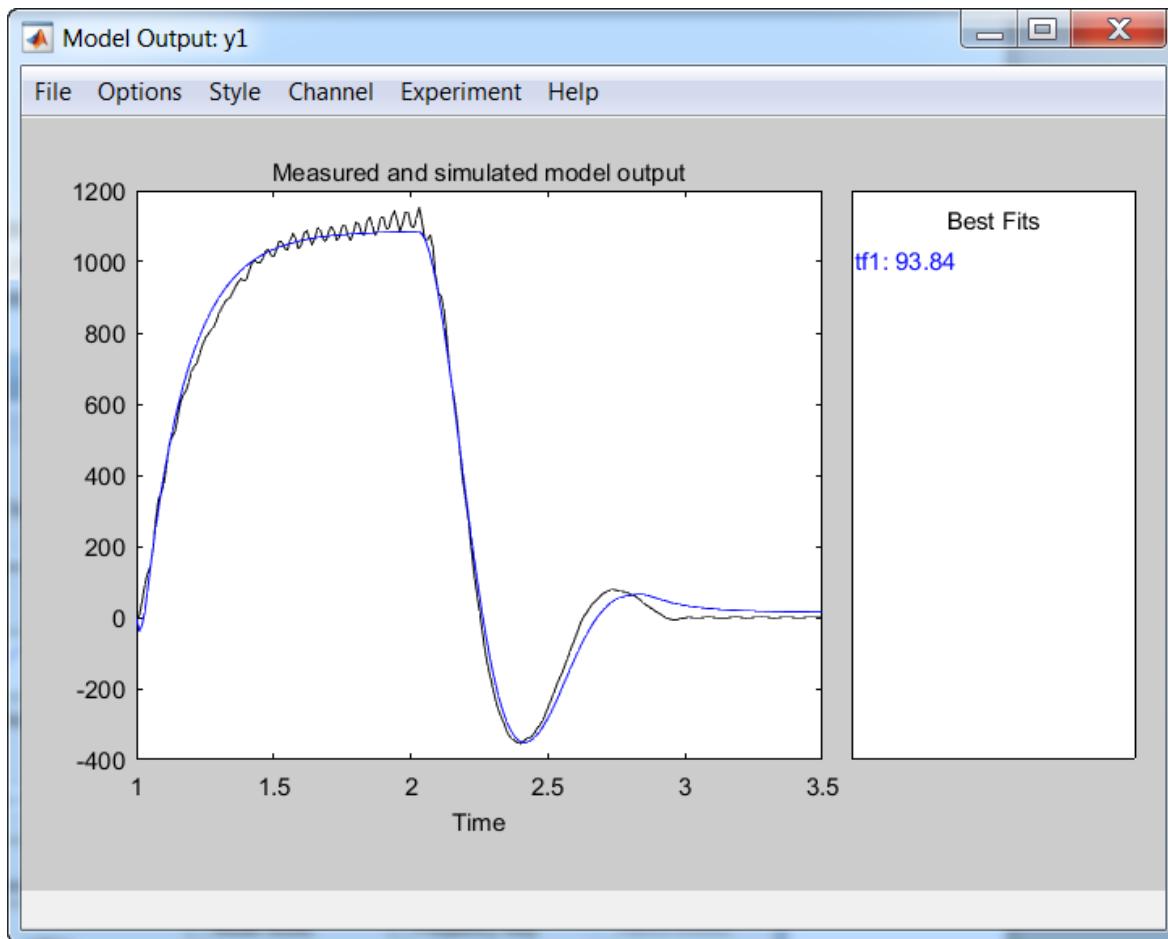


Figure 353: Overlap of the data from the experiment and the model resulting from the identification

C. Using the command line method

To identify the system response by using command lines, we first need to create an *iddata* type object containing all the elements necessary for identification:

- The vector containing the system response
- The vector containing system input
- Sampling period of data

Type the following command:

```
>> mydata=iddata(v,u,0.01)
mydata =
```

Time domain data set with 251 samples.
Sample time: 0.01 seconds

Outputs Unit (if specified)
y1

Inputs Unit (if specified)
u1

Now simply request an identification by transfer function by specifying the number of poles and the number of zeros in the transfer function. This is done using the *tftest* command by specifying 2 poles and 0 zero just as was done using the identification toolbox.

```
>>H = tftest(mydata,2,0)

H =

From input "u1" to output "y1":
5.746e05
-----
s^2 + 3218 s + 2.119e04
```

Continuous-time identified transfer function.

Parameterization:

Number of poles: 2 Number of zeros: 0

Number of free coefficients: 3

Use "tfdata", "getpvec", "getcov" for parameters and their uncertainties.

Status:

Estimated using TFEST on time domain data "mydata".

Fit to estimation data: 93.84% (simulation focus)

FPE: 1040, MSE: 999

MATLAB then returns the second order transfer function identical to that obtained using the identification toolbox.

Useful commands for identification	
Functions	Commands
Creating an <i>iddata</i> object for identification	$\text{mydata} = \text{iddata}(y, u, Ts)$ y: vector containing output data u: vector containing input data Ts: sampling period data
Identification of the system response by a transfer function	$H = \text{tfest}(\text{mydata}, np, nz)$ mydata: object created to aide the iddata function np: number of desired poles for the transfer function nz: number of desired zeros for the transfer function

There are also numerous commands that allow the preparation of data for identification (filtering, offset settings, replacement of missing data...). For more information on these commands, you can check the **System Identification Toolbox**.

Chapter 9: Control command with MATLAB - Simulink

I. Introduction

MATLAB-Simulink has very powerful system control and monitoring tools. The software can automatically perform the adjustment of a PID controller in an environment where the user acts on the set parameters without having to resort to theoretical bases. The software also offers expert modes where all the conventional system control methods are used and can be implemented. The main advantage is to have all the frequency and temporal plots required in the process of synthesizing a compensator. These plots change dynamically depending on the compensator structure imposed by the user.

II. Automatic PID settings

To illustrate the control system procedure with MATLAB and Simulink, we will study the control system with the speed of a DC motor.

A. Modeling

Electrical, mechanical and coupling equations of a DC motor are given

$$\begin{cases} u(t) = e(t) + R i(t) + L \frac{d i(t)}{dt} \\ e(t) = K_e \omega_m(t) \\ J \frac{d \omega_m(t)}{dt} = C_m(t) - C_r(t) - f \omega_m(t) \\ C_m(t) = K_t i(t) \end{cases}$$

u(t): motor supply voltage (V)
e(t): electromotive force (V)
i(t): armature current (A)
C_m(t): motor torque (N.m)
C_r(t): resistant torque (N.m)
ω_m(t): angular speed (rad/s)
R : armature resistor of the motor (Ω)
L : armature inductor of the motor (H)
J : inertia of the motor (kg.m²)
f : viscous friction parameter (N.m.s)
K_t : torque constant (N.m/A)
K_e : electromotive force constant (V.s/rad)

Figure 354: DC motor modeling

Applying the Laplace transform to these equations yields the block diagram of the DC motor. It controls the motor speed according to the block diagram in Figure 355.

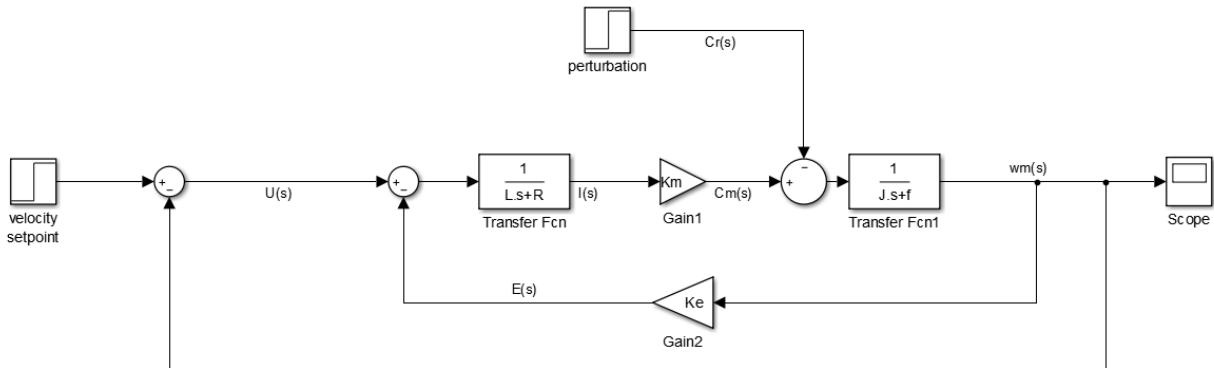


Figure 355: Speed control of a DC motor

B. Opening the model

Open the file ***motor_cc_controlled_velocity_US.slx***
Open the script ***motor_parameters.m*** and run it.

The file contains the modeling of the DC motor control of Figure 354.
Launch the simulation and view the response in the scope.

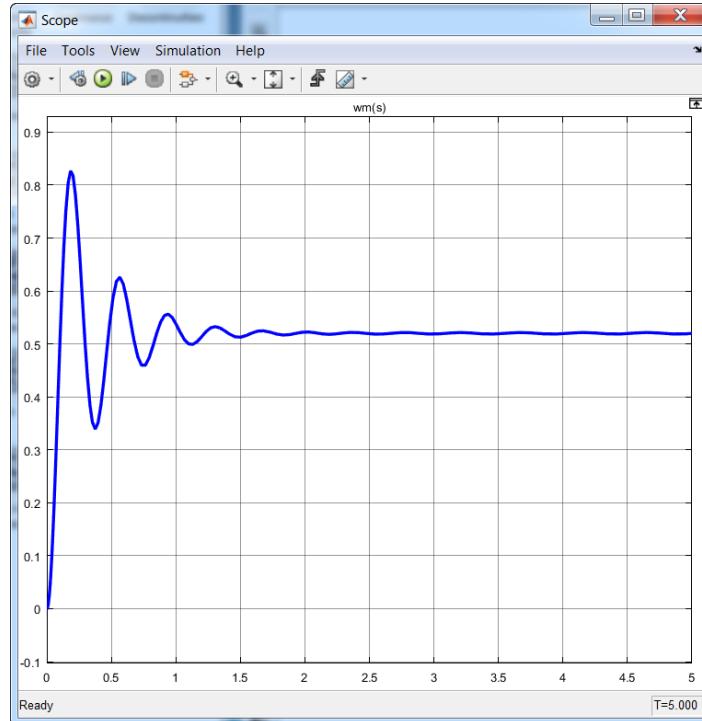


Figure 356: Time domain response of the uncorrected system at a unitary level

To perform the automatic PID setting, **insert** the **PID Controller** block according to Figure 357.

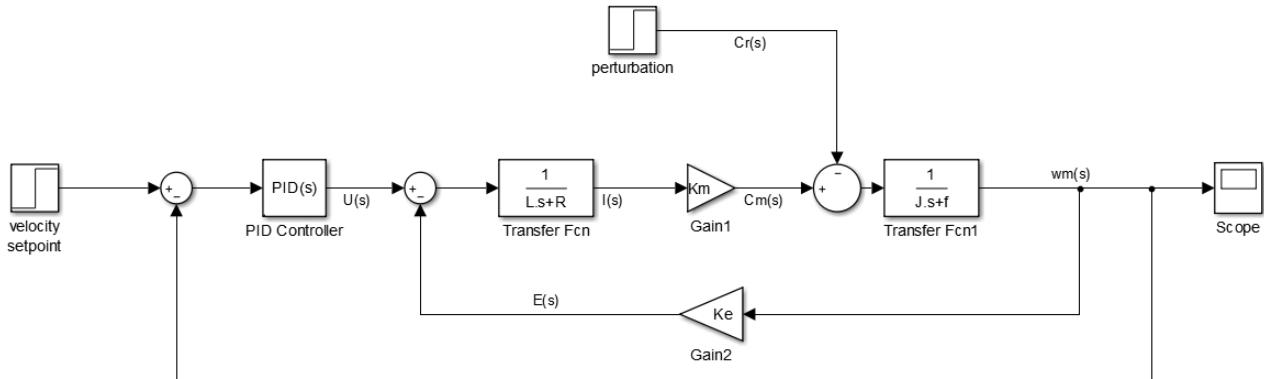
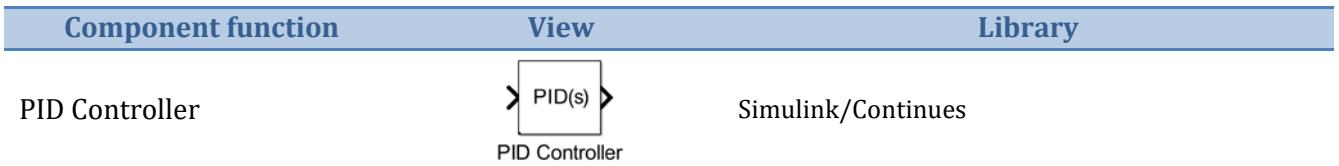


Figure 357: Adding a PID controller block into the control system

Double click on the **PID Controller** block to open the configuration window as in Figure 358.

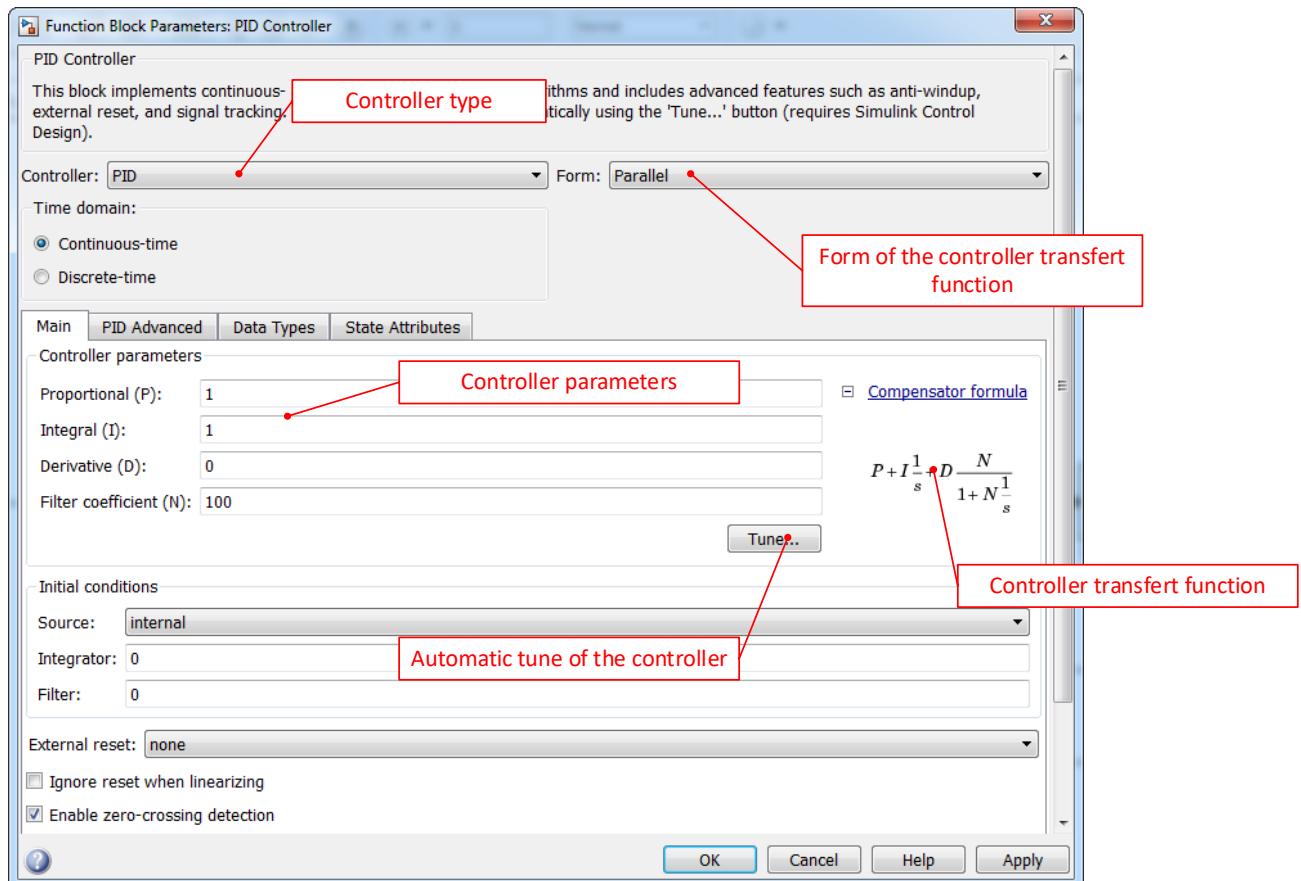


Figure 358: PID Controller configuration window

1. Analysis of the time domain response

Click on **Tune** to open the settings window for the system performances.

MATLAB offers a setting that achieves a compromise between all system performance criteria as shown in Figure 359. By default the step response of the closed-loop transfer function (reference tracking) is displayed for the corrected system and the uncorrected system.

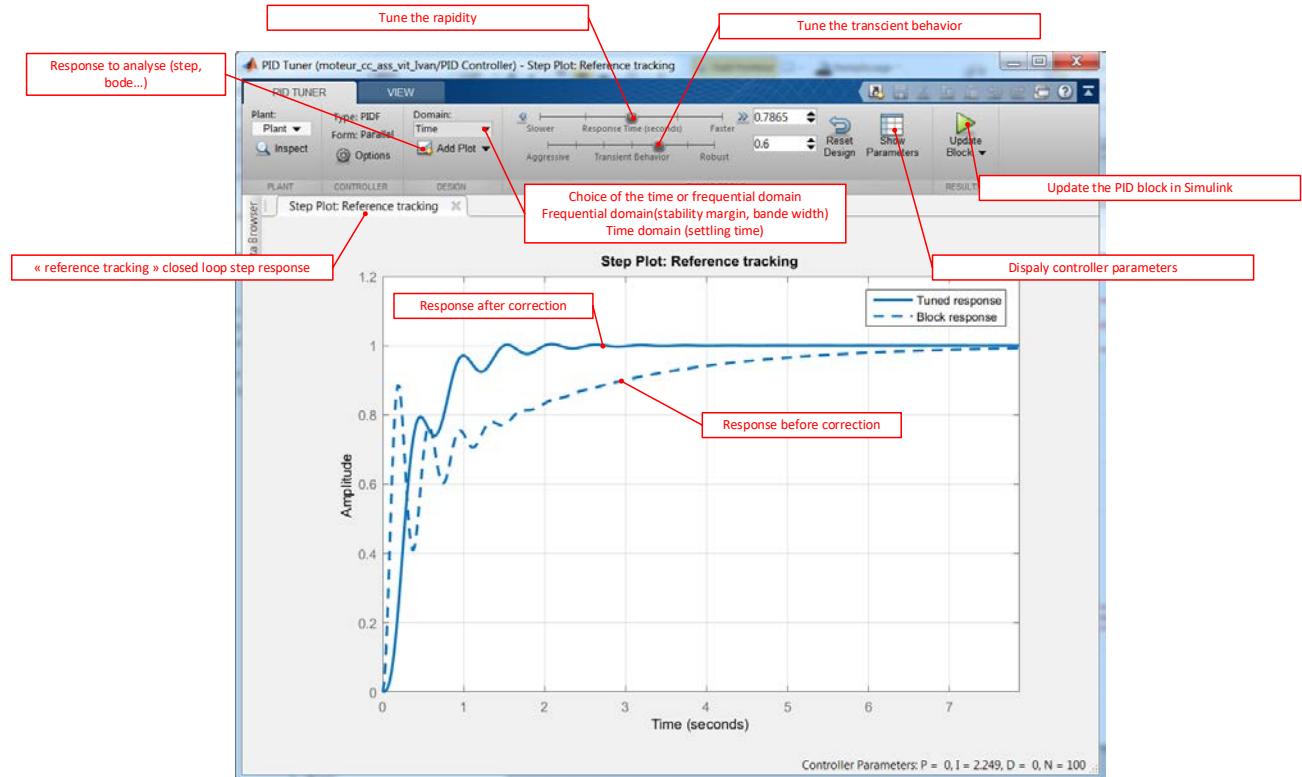


Figure 359: Settings window for system performances

However, it is possible to manually operate the cursors to change the system performances and make it conform to precise specifications. The tool offers different features to provide all the information needed to make the adjustments.

Click on **Show Parameters** at the top right of the window to view the performance criteria and the values of the correction gains before and after the correction as shown in Figure 360.

Controller Parameters		
	Tuned	Block
P	0	1
I	2.2488	1
D	0	0
N	100	100

Performance and Robustness		
	Tuned	Block
Rise time	0.719 seconds	2.95 seconds
Settling time	1.85 seconds	6.05 seconds
Overshoot	0.466 %	0 %
Peak	1	0.999
Gain margin	7.32 dB @ 11.8 rad/s	Inf dB @ Inf rad/s
Phase margin	83.8 deg @ 2.54 rad/s	33.7 deg @ 16.1 rad/s
Closed-loop stability	Stable	Stable

Figure 360: System performance criteria and compensator gain values

To get the **Open-loop transfer function Bode diagram**, select **Add Plots/Bode/Open Loop**.

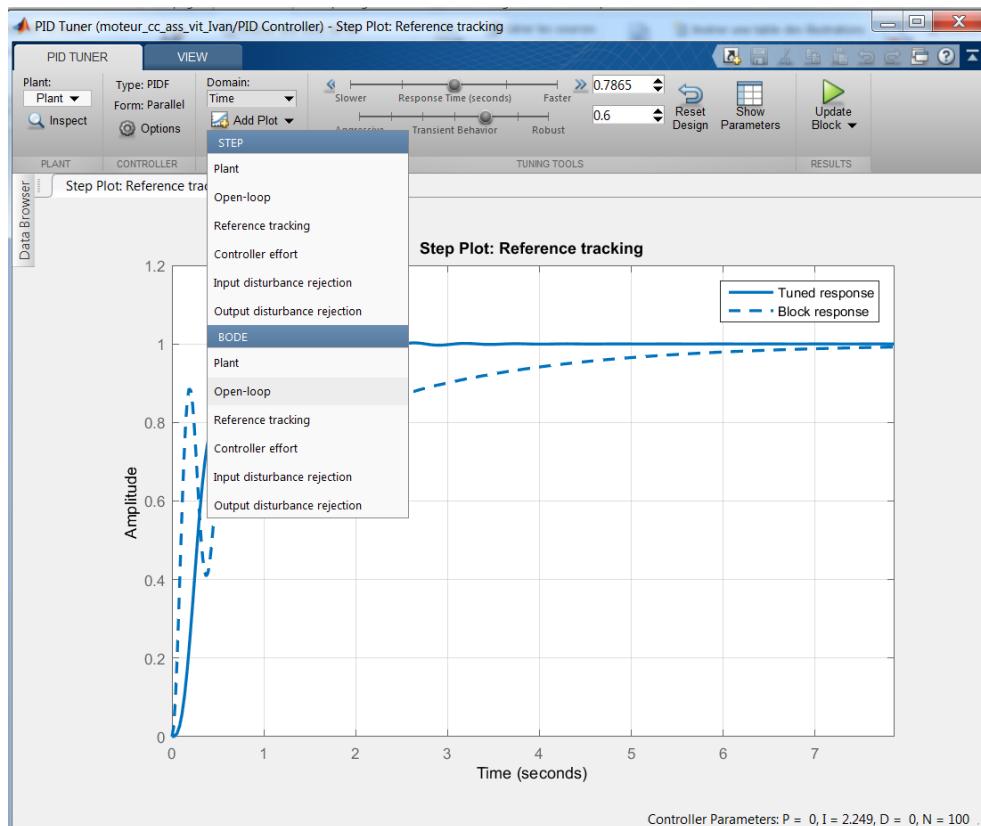


Figure 361: Adding a Bode diagram to the open-loop transfer function

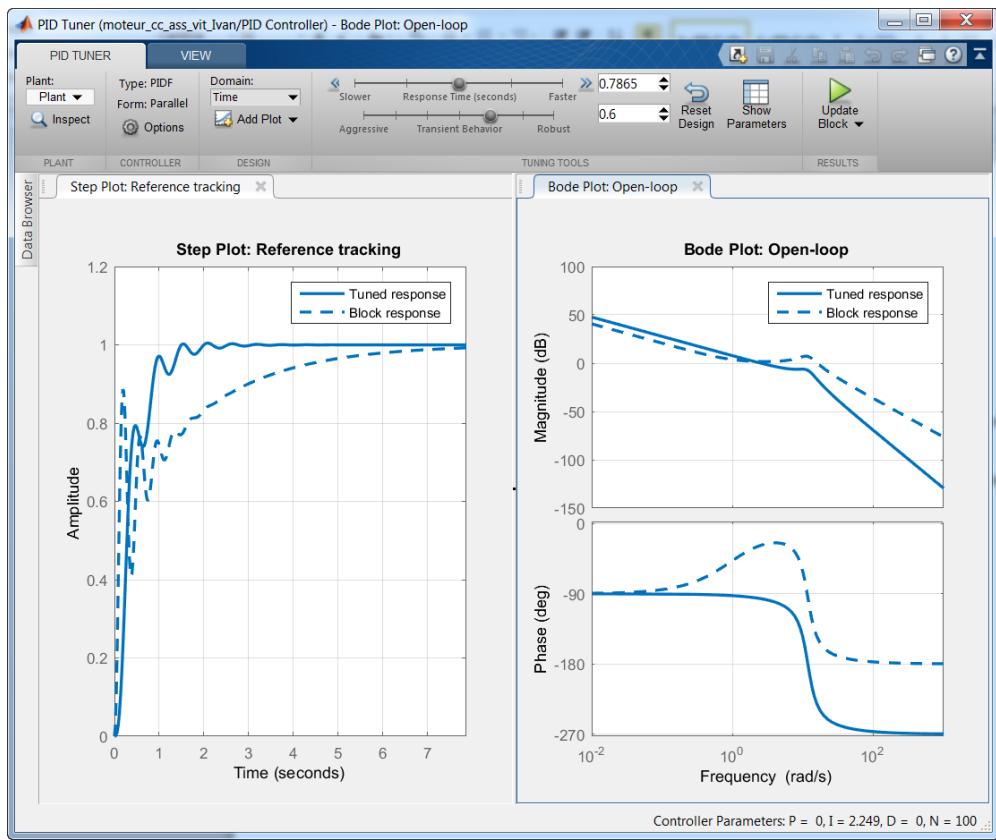


Figure 362: Display of the Bode diagram of the open-loop transfer function for PID settings

Bode diagrams for corrected and non-corrected systems will appear in a second window.

To set the performance criteria options, **right-click** in the time domain response graphics window then select **Properties**, then **Options** tab.

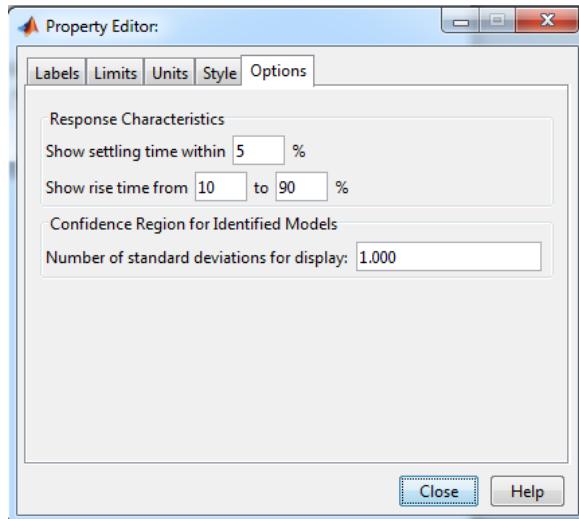


Figure 363: Setting the performance criteria options

Indicate that the response time is measured at 5% response time indicating 5% in the field **Show settling time within**. The **rise time** criterion deals with the rise time, which is estimated between 10% and 90% of the final value reached. Click **Close**.

It is also possible to graphically display all the performance criteria.

Right-click in the time domain response graphics window then select **Characteristics/Peak Response** (passing), then **Characteristics/Settling Time** (response time of 5%), then **Characteristics/Rise Time** (rise time) then **Characteristics/Steady State** (accuracy).

Right-click in the Bode diagram graphics window, then select **Characteristics/All Stability Margins** (gain margin and phase margin).

It is now possible to view the performance criteria in the graphics window.

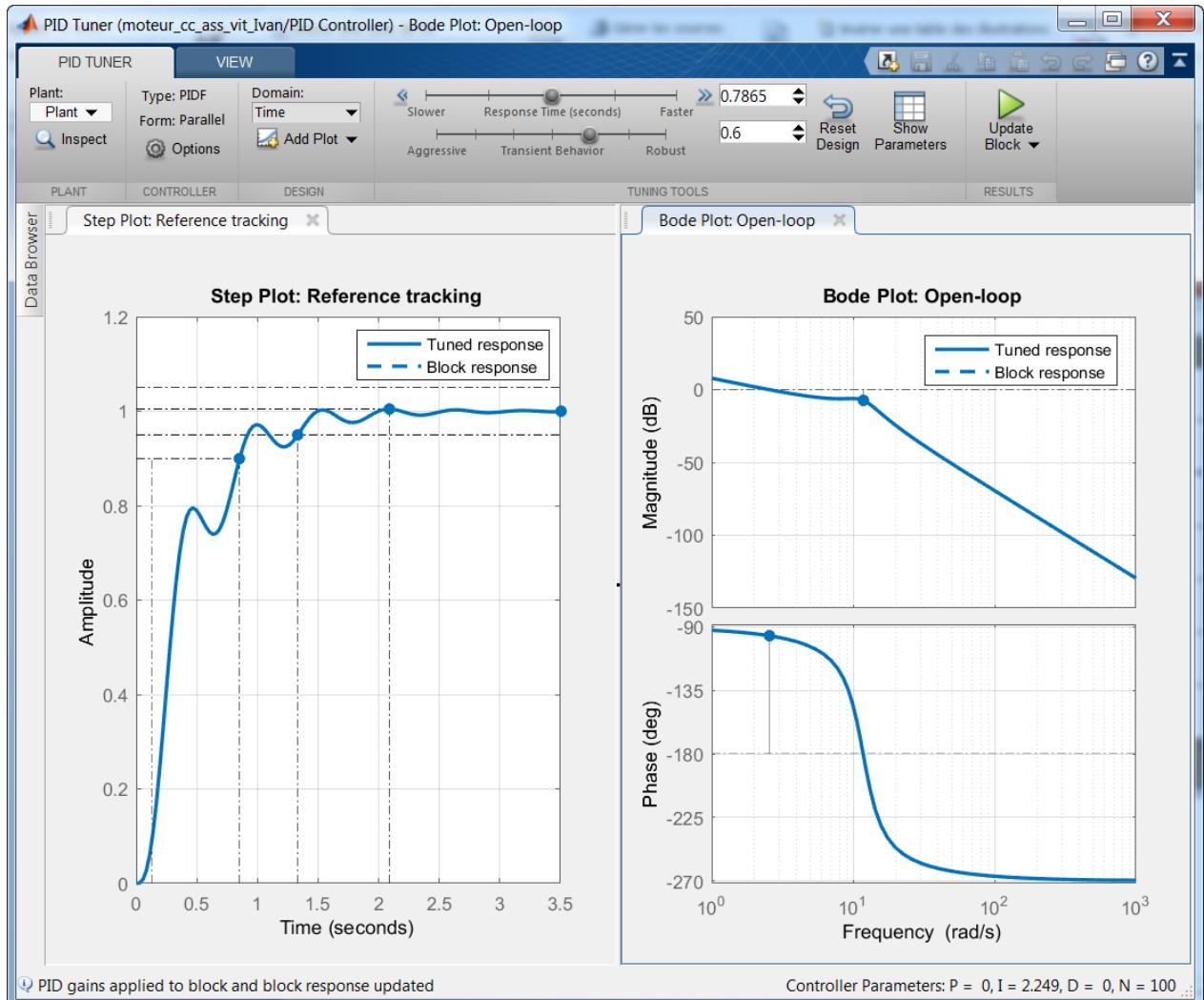


Figure 364: Viewing performance criteria

At this stage, it is possible to move the cursors and see the curves and parameters change interactively.

Once the settings have been configured, **Check** the box **Update Block** to automatically update the **PID Controller** block of **Simulink**.

2. Importing into Simulink

Once the system response is correctly adjusted, click OK to return to the PID parameters block. We can see that the gains from the compensator have been updated with the values corresponding the adjustment that has been made.

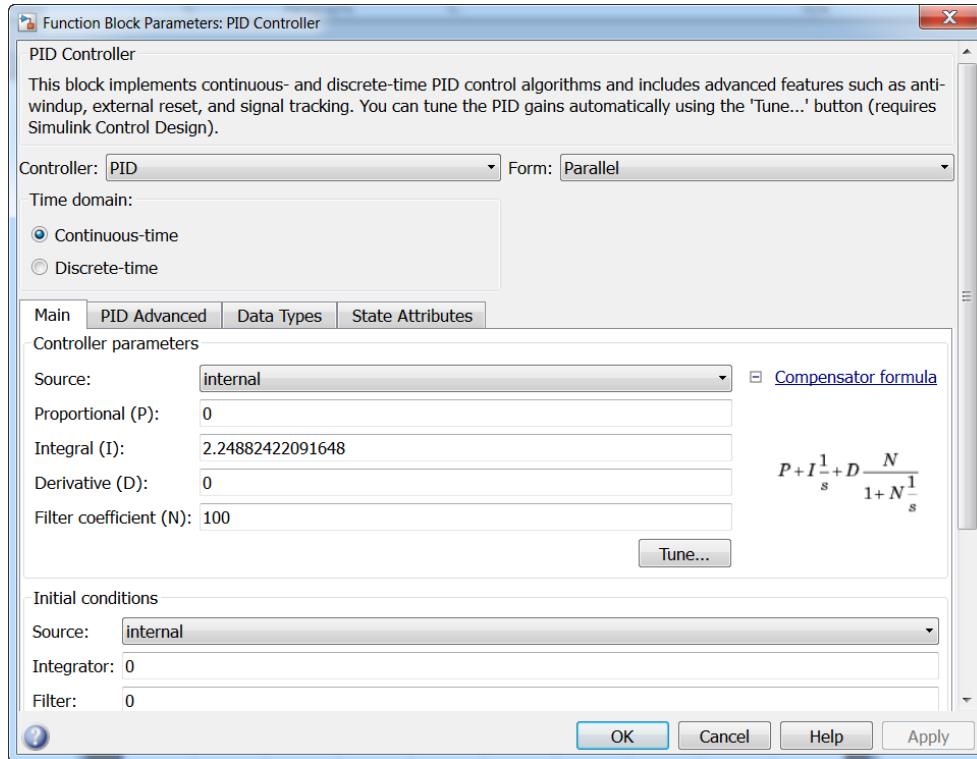


Figure 365: Automatically updating the PID parameters

Click OK.

Launch the simulation and view the corrected response in the scope.

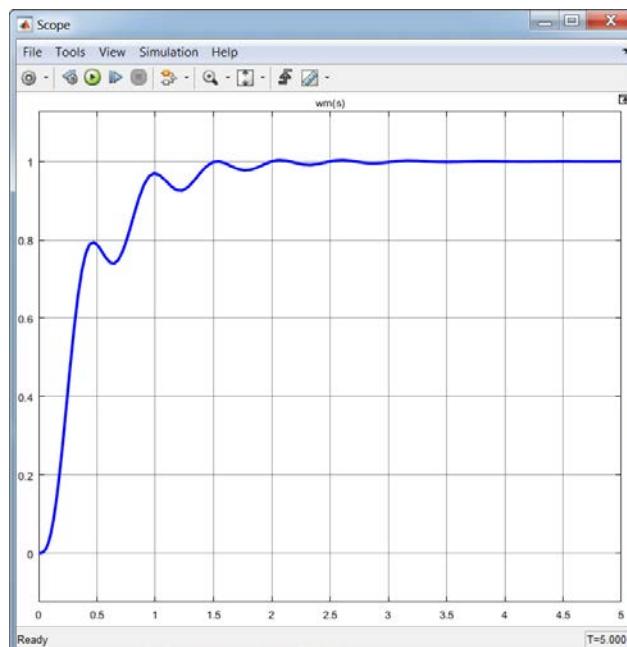


Figure 366: System response after PID adjustments

III. Manual adjustments to a PID with the “compensator design” tool

It is also possible to set a PID while viewing the influence of adjustments to the time domain and frequency responses of the open-loop transfer function and the closed-loop transfer function. To do this, you must use the **Simulink** “compensator design” tool.

A. Opening the model

Open the file **moteur_cc_ass_vit_PID_comp_des.slx**.

This file contains the DC motor speed control with a PID that has not yet been set (proportional gain at 1 and other gains are zero).

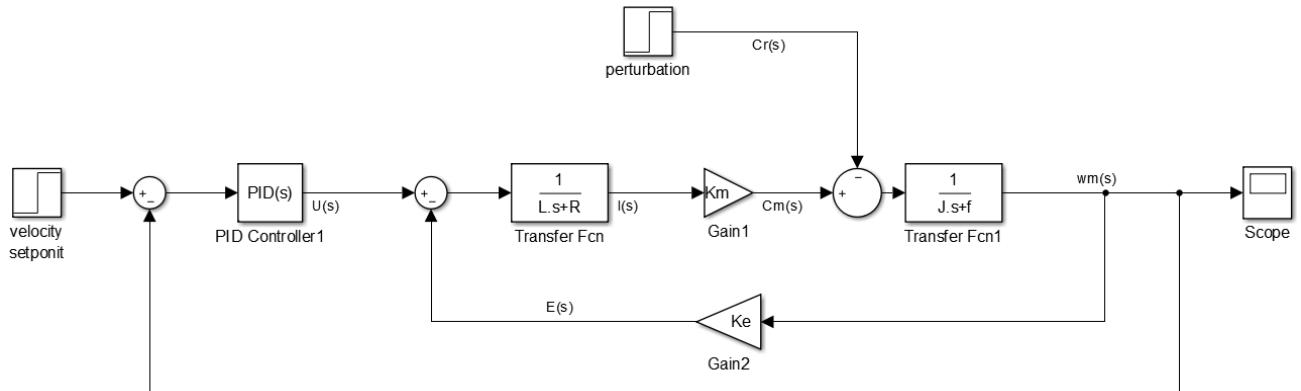


Figure 367: Speed controlled DC motor model with disturbance

The imposed guideline is 100 rad/s and a disturbance torque is applied to the system at time t=3s.

The performance criteria are as follows:

- Steady state error equals to zero
- rise time (from 0 to 90%) of 0.8s.
- settling time 5% <1s
- maximum overtaking 10%
- Gain margin: 20dB
- Phase margin 45°
- rejection of steady disturbance

Launch the simulation and observe the system response in the scope.

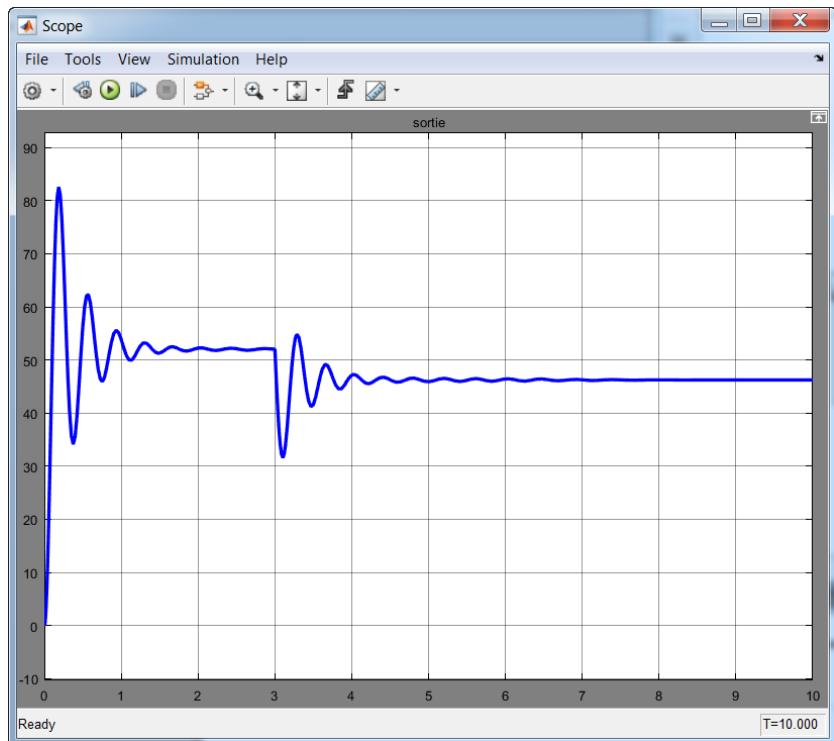


Figure 368: Time domain response of the uncorrected system

We can see that the system response is not satisfactory:

The system is not accurate, poorly damped and does not correctly reject the disturbance.

B. Adjusting the PID

In order to use the “compensator design” tool, it is necessary to place linearization points on our model. The input and output of the **closed-loop transfer function** must be indicated so that the “compensator design” tool can be used. **MATLAB** will automatically detect the **closed-loop transfer function** and can trace the locations of open-loop transfers utilized to make adjustments.

1. Placement of linearization points

Place the following linearization points (placement of the linearization points is described in detail on page 234):

- an **Input Perturbation** type linearization point on the signal that represents the speed set point
- an **Input Perturbation** type linearization point on the signal that represents the disturbance
- an **Output Measurement** type linearization point on the signal that represents the output speed of the motor

You need the model of Figure 369.

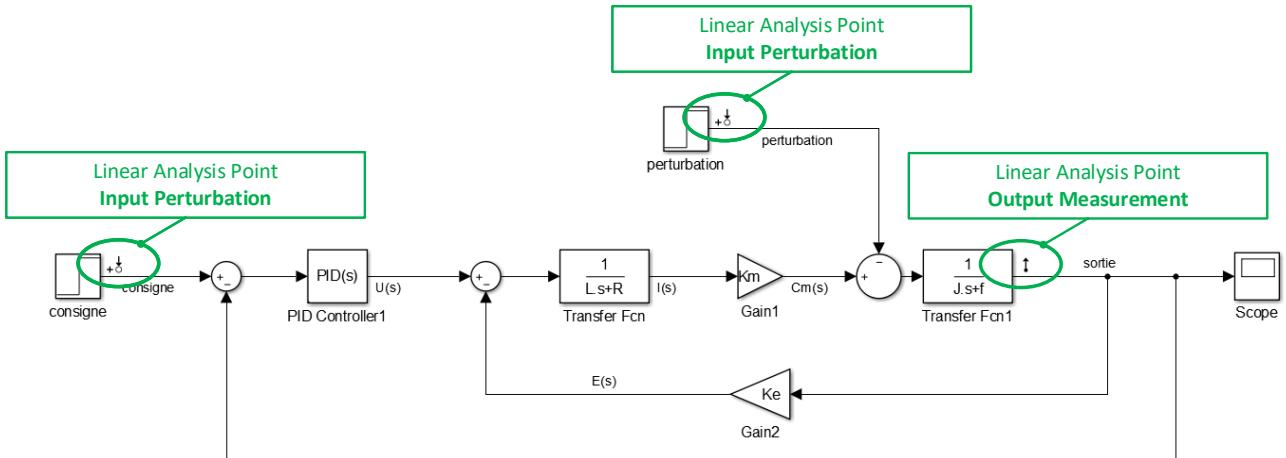


Figure 369: Placing linearization points on the model

From the command bar, select **Analysis/Control Design/Control System Designer**

This command will open the **Control and Estimation Tools Manager** window that lets you choose the blocks you want to adjust and view the input and output points of the closed-loop.

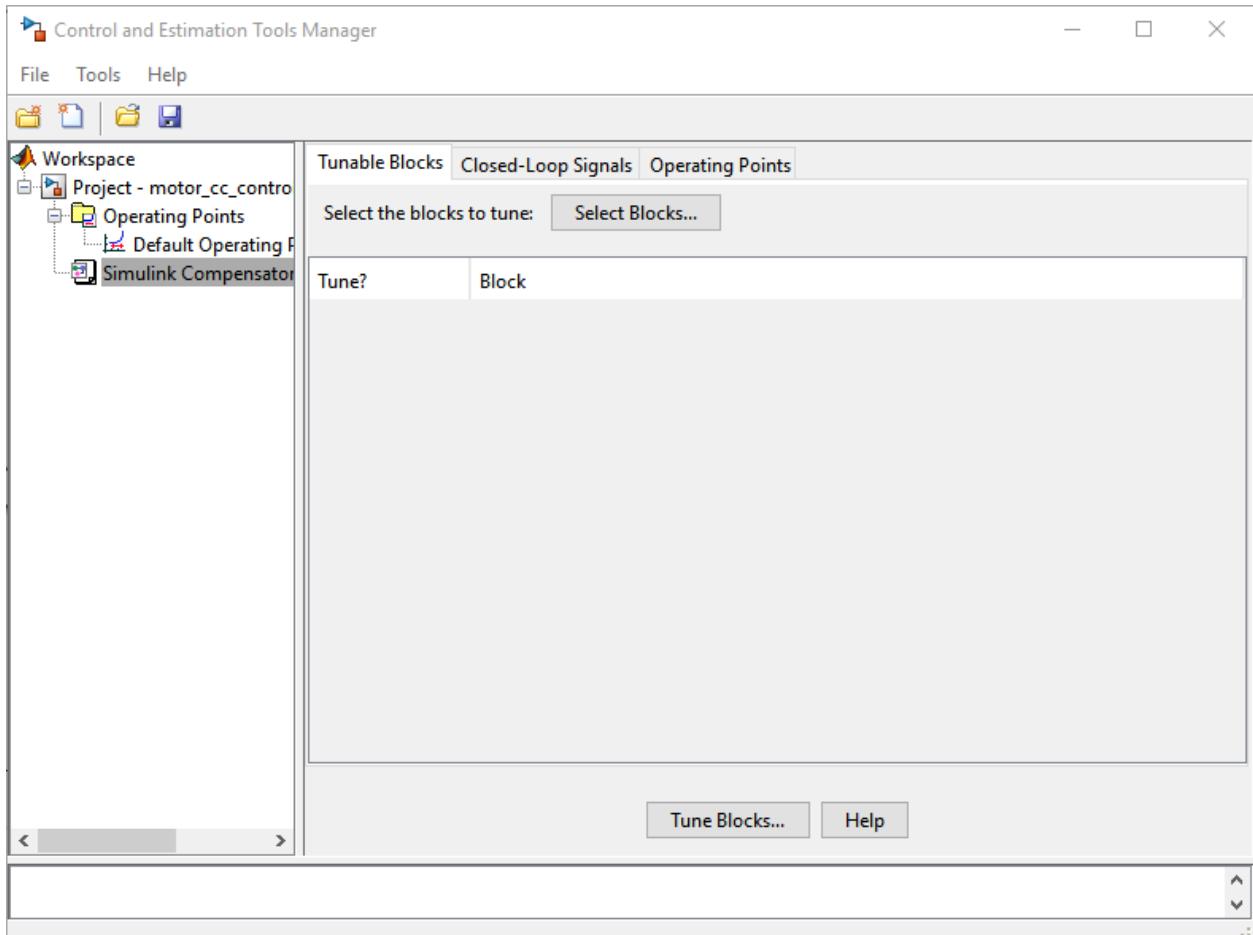


Figure 370: Control and Estimation Tools Manager window

Select the **Closed-Loop signal** tab to view the input and output signals taken into account.

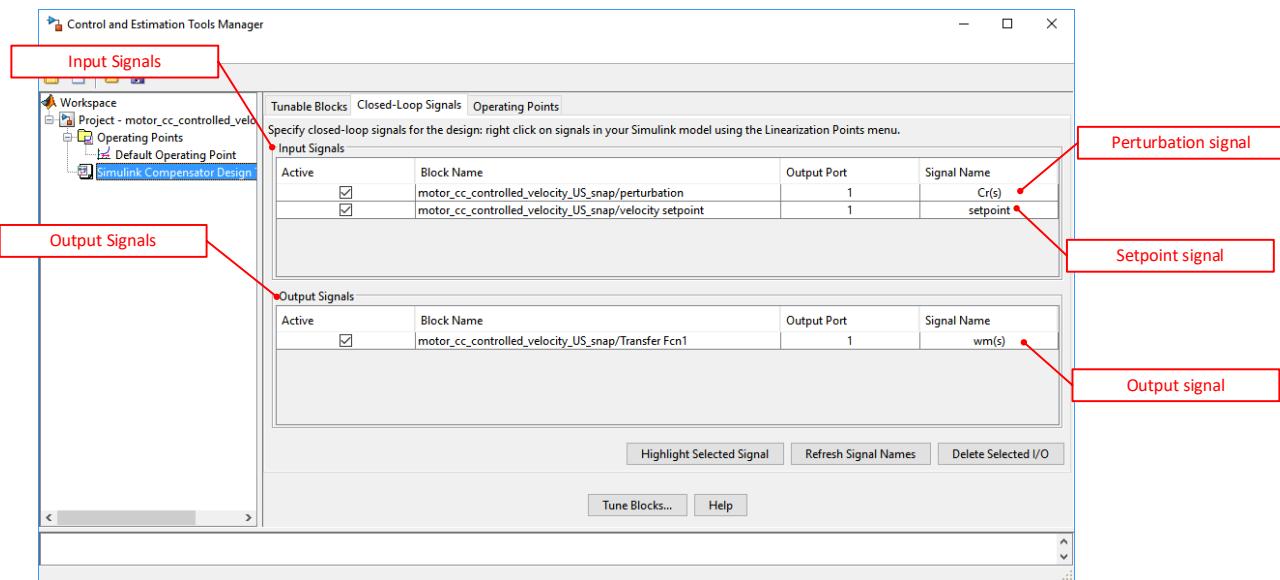


Figure 371: Viewing the input and output signals taken into account for the closed-loop

It will then be possible to construct all possible transfer functions from these signals.

It is possible to select a signal and highlight it in the **Simulink** model by clicking **Highlight Selected Signal**, which provides control of input and output points of the closed-loop.

2. Choosing a block to adjust

Select the **Tunable Blocks** tab to choose a block to adjust then click **Select Blocks**.

The **Select Blocks to Tune** window appears and presents all the blocks of the model that can be adjusted (gains, transfer functions, PIDs...).

Select **PID Controllers** and click **OK** to adjust the PID.

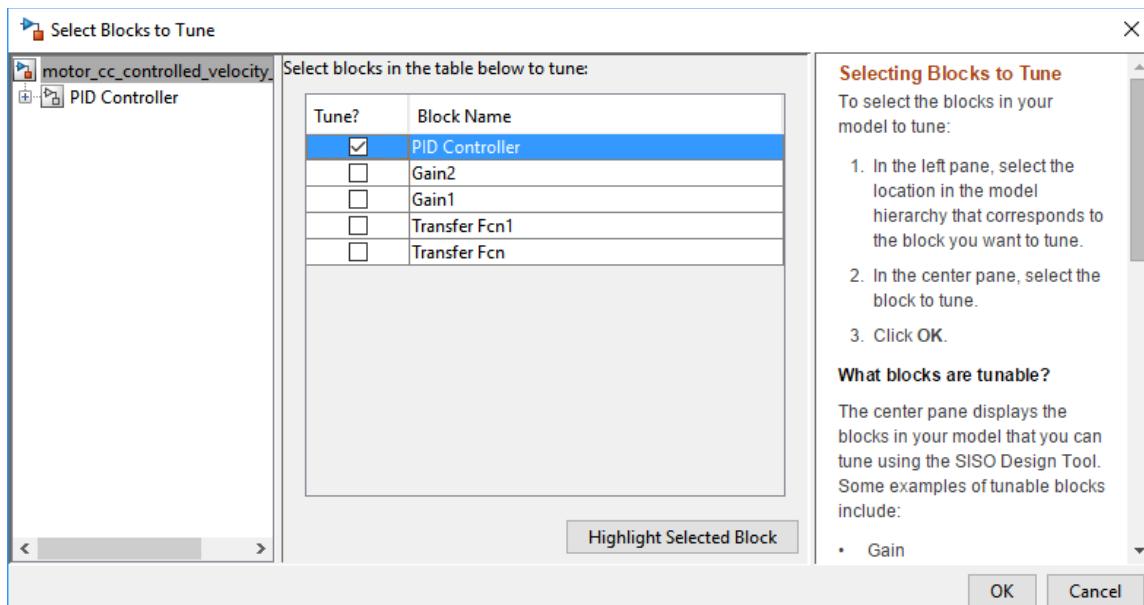


Figure 372: Choosing a block to adjust

Then click on **Tune Block** to begin the adjustments.

An information window appears indicating that you will open the **Siso Design Tool**. This window tells you that you can work in this tool with two types of plots:

Design plots: these pole placement plot types, Bode diagram or Black-Nichols diagram of the open-loop transfer function are interactive in the tool. They will be used to graphically adjust system performances by using adjustment methods based either on analysis of the open-loop transfer function, or on the analysis of the closed-loop transfer function. On these plots it will be possible to modify the compensator gains by manually moving the transfer locations or by adding and removing the poles and zeros in the transfer function of the compensator...The plots that you want to appear are defined at the beginning of the use phase of the tool but can be changed at any time during the study.

Analysis plots: these step response type plots, Bode diagram of the closed loop transfer function etc., show the effects of the settings on the system performance.

To summarize, we must do and change the **Design Plots** and observe the effects on the **Analysis Plots**. Before choosing the different plots, it is important to thoroughly analyze the desired approach and display only the relevant plots.

Click **Next**.

3. Choosing plots to display for the open-loop

The Design Configuration Wizard window opens and lets you choose **Design Plots** type plots which can be changed to make adjustments. (SISOTOOL Design Views)

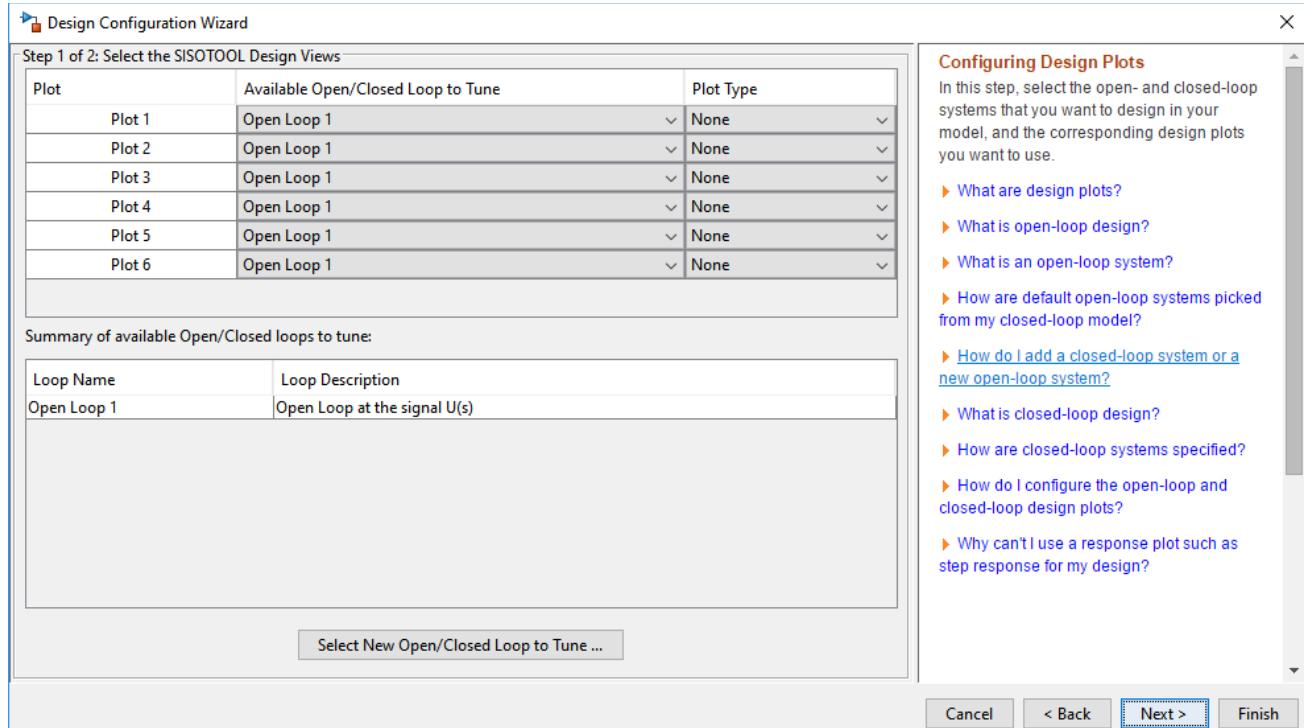


Figure 373: Design Configuration Wizard window

Using the drop-down menus we can choose: (Figure 373)

- Root locus (locus of the poles in a **closed-loop transfer function**)
- Bode diagram of the **open-loop transfer function**

- Nichols diagram of the **open-loop transfer function**

Note that in this window all plots are related to the open loop transfer function. Even the location of the poles of the closed-loop transfer function is determined by the variation of the gain of the considered open-loop transfer function (Evans locus)

These curves will help us visualize parameters such as margins of gain and phase, the position of the poles of the closed-loop transfer function that will provide information on the stability or on the depreciation, etc..

We will have the opportunity to make changes to these plots to change the parameters of the compensator and proceed with adjustments.

Configuring **Design Plots** type plots as indicated in Figure 374.

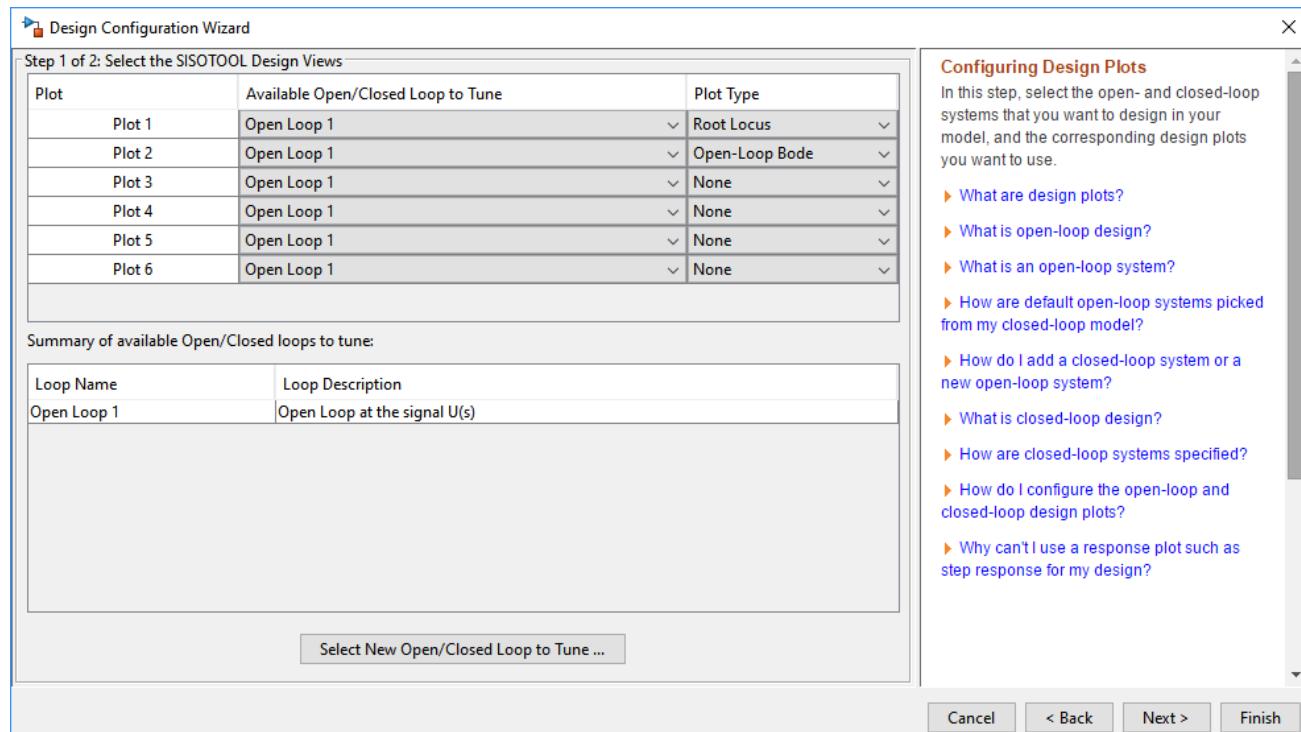


Figure 374: Choosing plots to view to adjust the PID

4. Choosing plots to view the performance of the closed-loop

The next window allows the viewing of **Analysis Plots** plot types, showing the performance of the closed-loop. These plots will be the effect of the adjustments made to the **SISOTOOL Design Views** plot types that deal with the open-loop transfer function. These plots cannot be changed and they are only available for viewing.

Using the drop-down menus we can choose: (Figure 375) different types of diagrams concerning all behaviors of the closed-loop transfer function:

- Step response
- Impulse response
- Bode diagram
- Black-Nichols diagram...

In our case we will use the following **Analysis Plots**:

- The time domain response of the **closed-loop transfer function** to a **set point step**
- The time domain response of the **open-loop transfer function** to a **perturbation step**

This configuration corresponds to Figure 374.

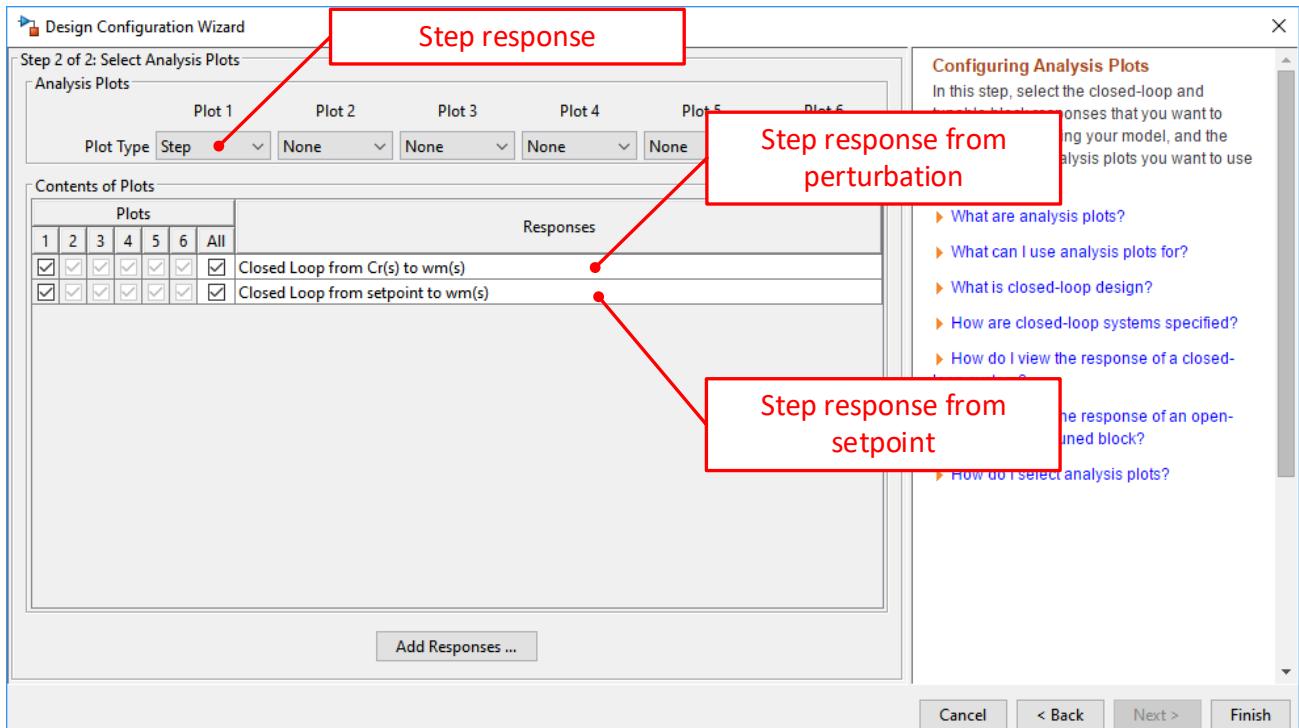


Figure 375: Choosing closed-loop responses

Click Finish

5. Analyze the graphic windows of the “Control System Designer” window

Two new windows will then appear on the screen:

Linear Analysis for SISO Design Task: this window shows the step responses of the closed-loop transfer function compared to the setpoint and compared to the perturbation.

To obtain the grid pattern of the graph, **Right-Click** the mouse in the graph window and select **Grid**.

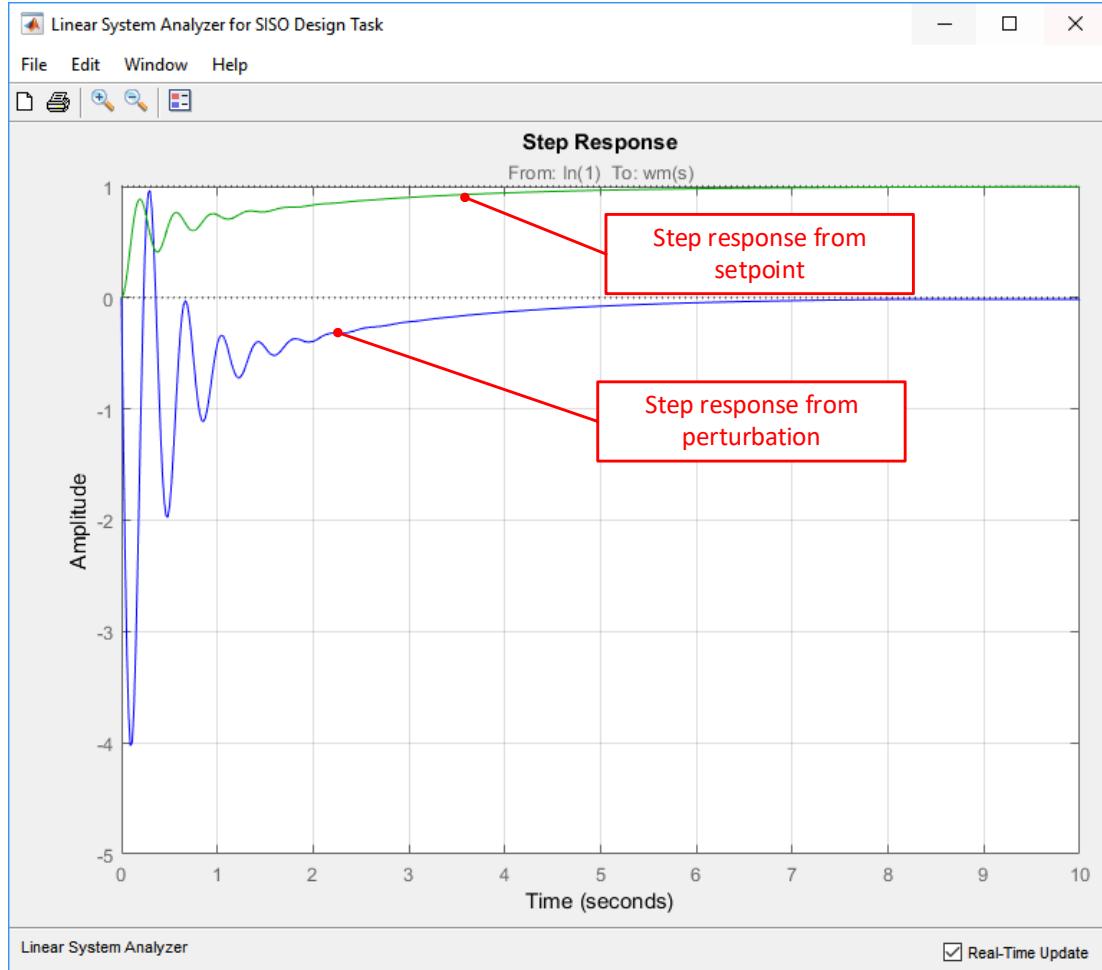


Figure 376: LTI Viewer for SISO Design Task

SISO Design for SISO Design Task: this window shows the plots that we can change to make adjustments to the PID.

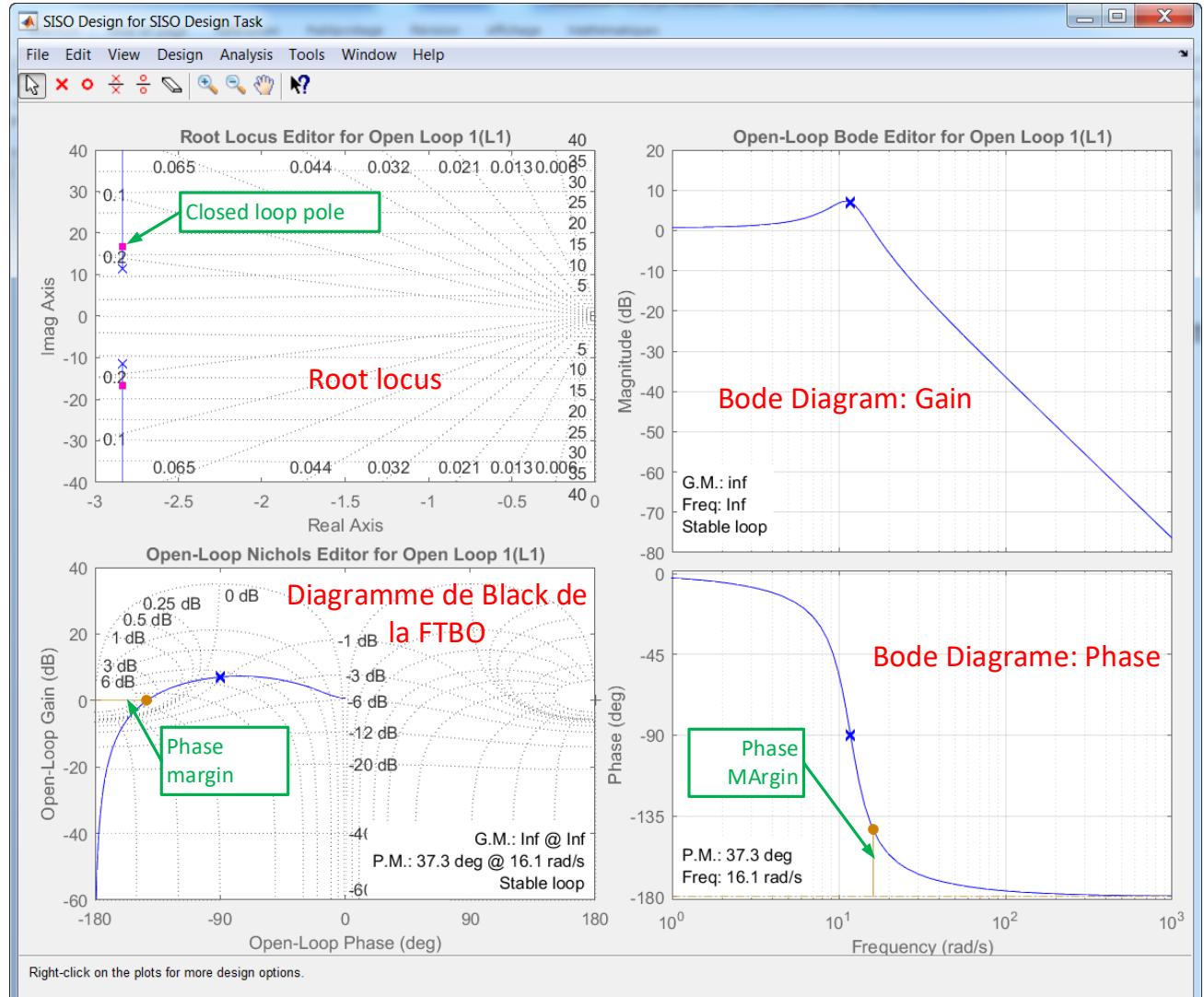


Figure 377: SISO Design for SISO Design Task

Note that the **Control and Estimation Tool Manager** window also stays open. This lets you return to the plot choices at any time by using the **Graphical Tuning** tab to change the plots of the **SISO Design for SISO Design Task** window and the **Analysis Plots** tab to change the plots of the **LTI View for SISO Design Task** window.

Adjustments will initially only be made using the step response of the closed-loop transfer function in relation to the set point.

In the **Control and Estimation Tool Manager** window select the **Analysis Plots** tab and deselect the box corresponding to the step response vis-à-vis the disturbance as indicated in

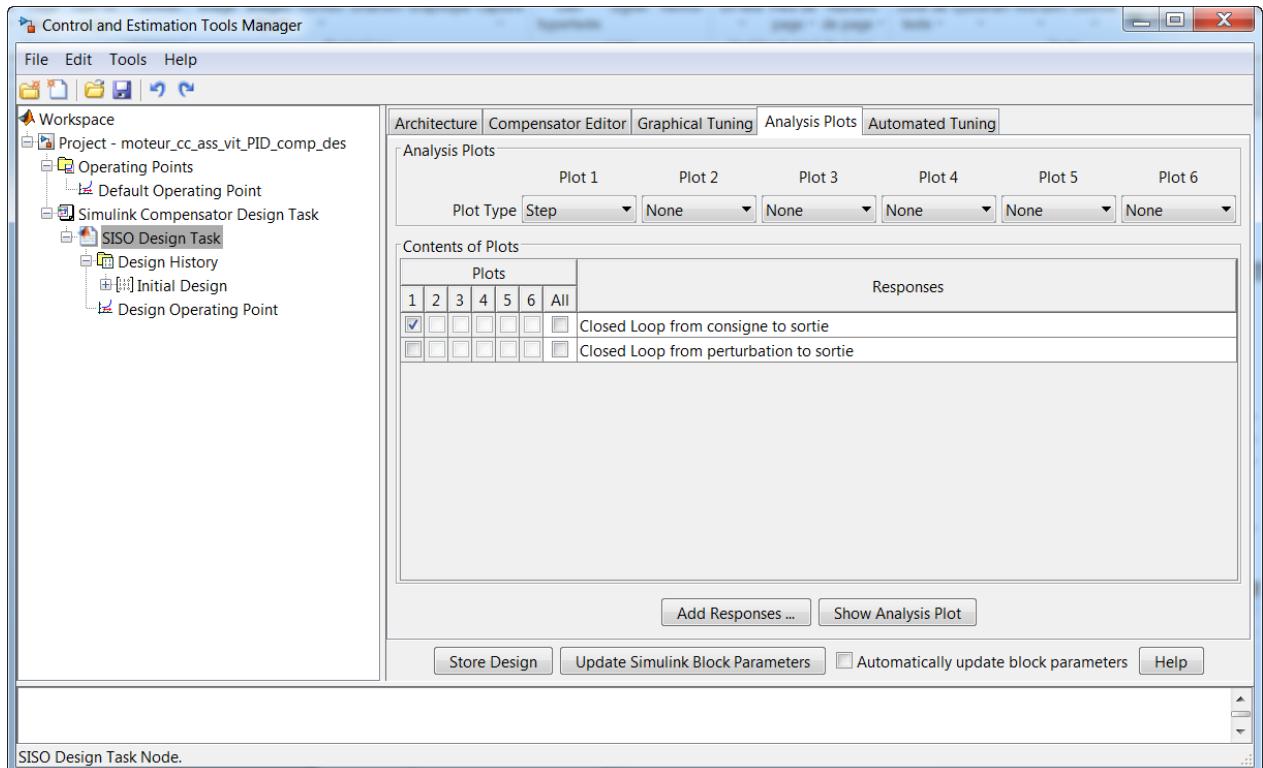


Figure 378: Modification of Analysis Plots choices

The **Linear Analysis for SISO Design Task** window is updated according to this new configuration (

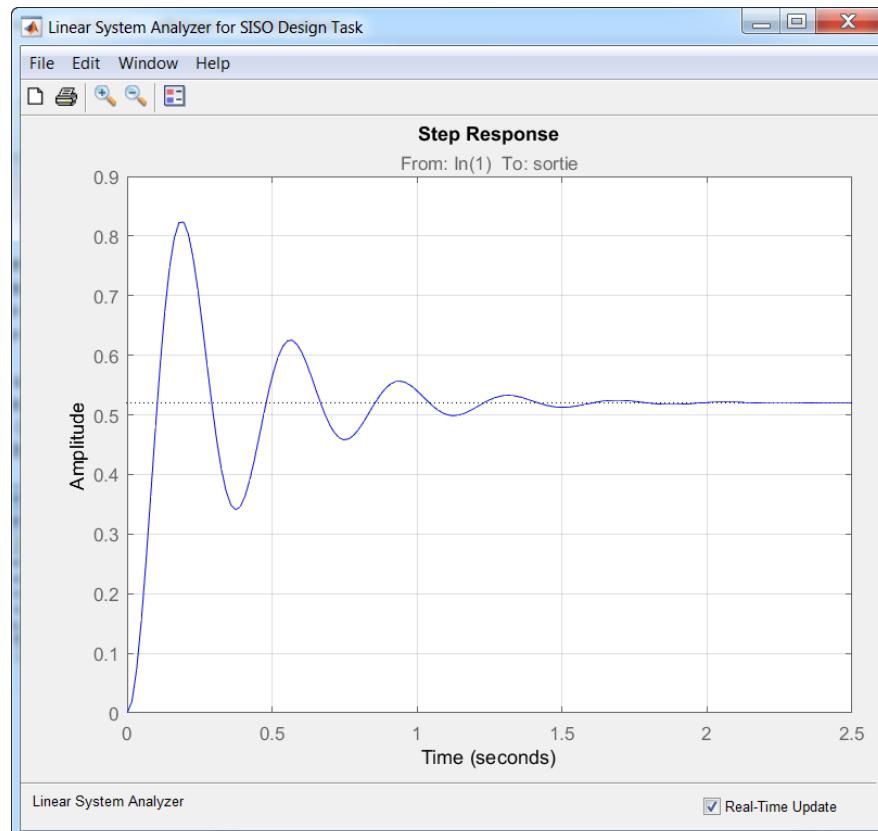


Figure 379: Updating the Linear Analysis window for SISO Design Task

In the **Control and Estimation Tool Manager** window select the **Compensator Editor** tab (Figure 380). This window allows the PID parameters to be adjusted and displayed.

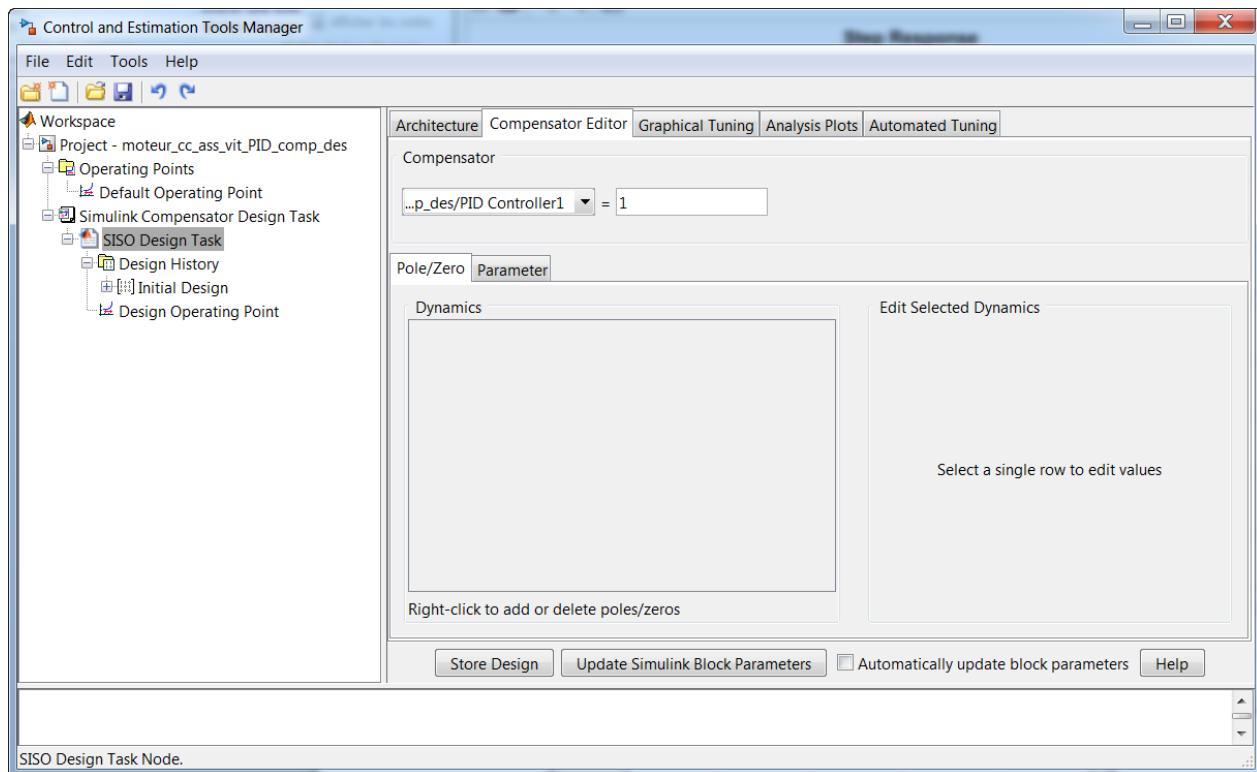


Figure 380: The Compensator Editor tab of the Control and Estimation Manager window

To make best use of this tool, it is necessary to be able to see these three windows simultaneously:

- **Control and estimation Tool Manager** to make adjustments
- **SISO Design for SISO Design Task** to see the influence of the adjustments on the behavior of the open-loop
- **Linear Analysis for SISO Design Task** to see the influence of the adjustments on the behavior of the closed-loop.

It would be best to have a large screen or to work with two screens.

6. Adjusting the PID

In the **Control and Estimation Tools Manager** window, **Compensator design** tab, click on the **Parameter** tab.

It is now possible to proceed with the PID adjustments and see all plots change dynamically.

Slide the different cursors and see the influence of the adjustments on the **Design Plots** and on the **Analysis Plots**.

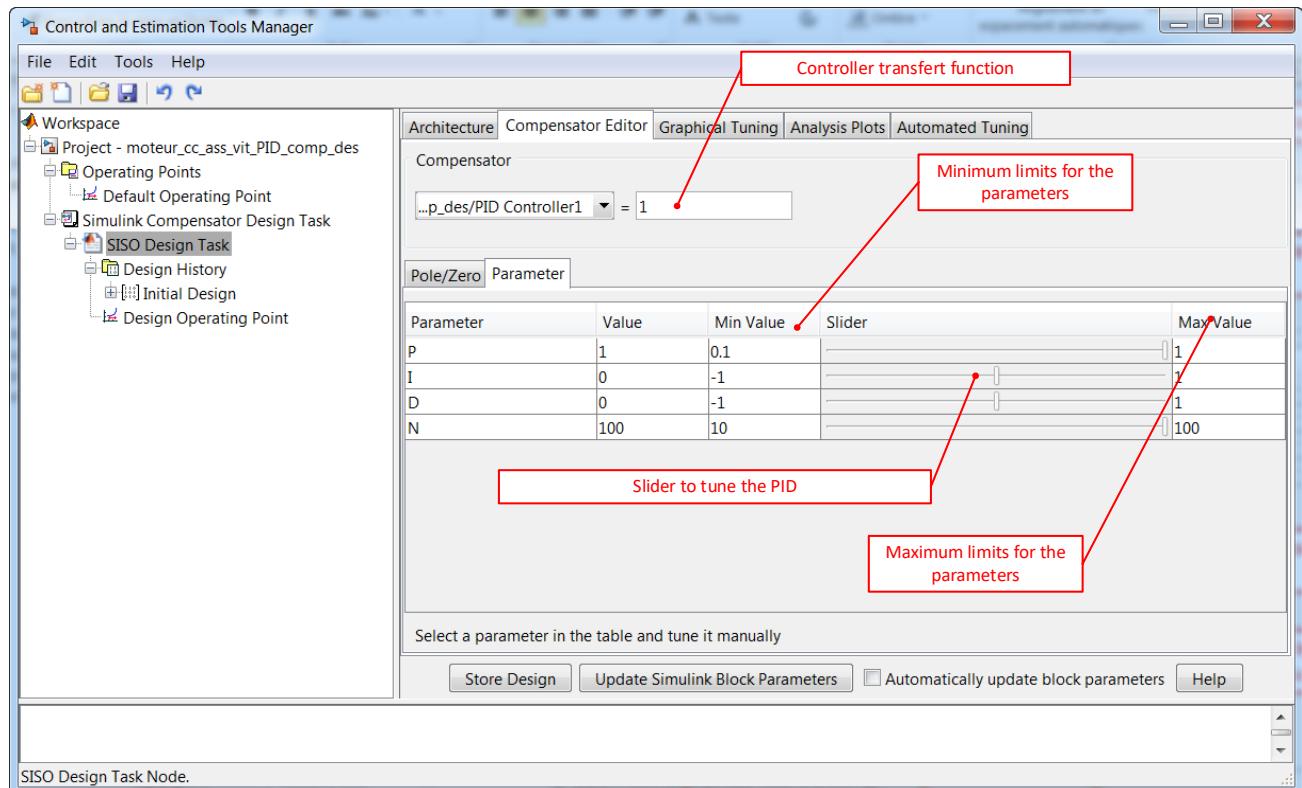


Figure 381: Functions of the PID adjustment window

7. Definition and display of performance criteria

Adjustments to the compensator can only be made if performance criteria are defined. It is possible to define and display these criteria in the corner of the **Linear Analysis for SISO Design Task** window.

Right-click in the graphic window showing the time domain response of the FTBF (closed-loop transfer function) and choose **Properties**, **Options** tab then define the speed criterion at 5% response time.

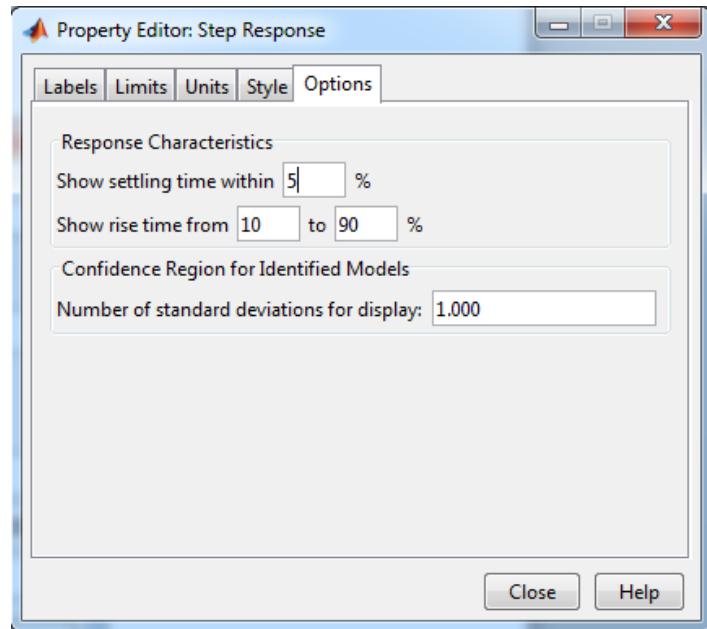


Figure 382: Definition of speed criteria

Right-click in the graphic window showing the time domain response FTBF (closed-loop transfer function) and choose **Characteristics** then **Settling Time** to set the response time to 5%.

Right-click in the graphic window showing the time domain response FTBF (closed-loop transfer function) and choose **Design Requirement New** and enter the performance criteria for the adjustment to the PID according to Figure 383.

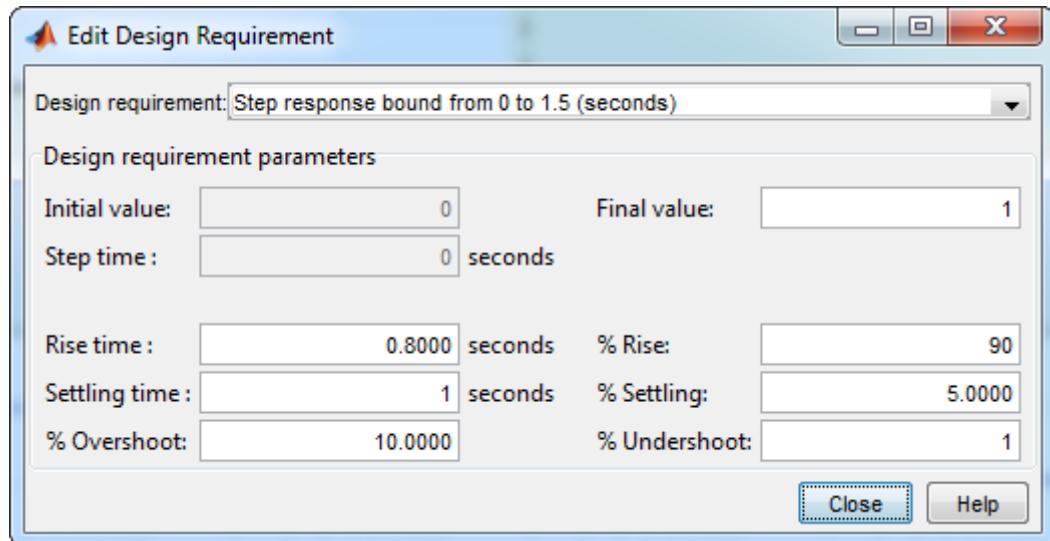


Figure 383: Definition of performance criteria

Here we impose:

- a rise time (from 0 to 90%) of 0.8s
- a response time at 5% of 1s
- a maximum load of 10%

The graphic window allows you to view the area in which the step response must be to meet the specifications (Figure 384).

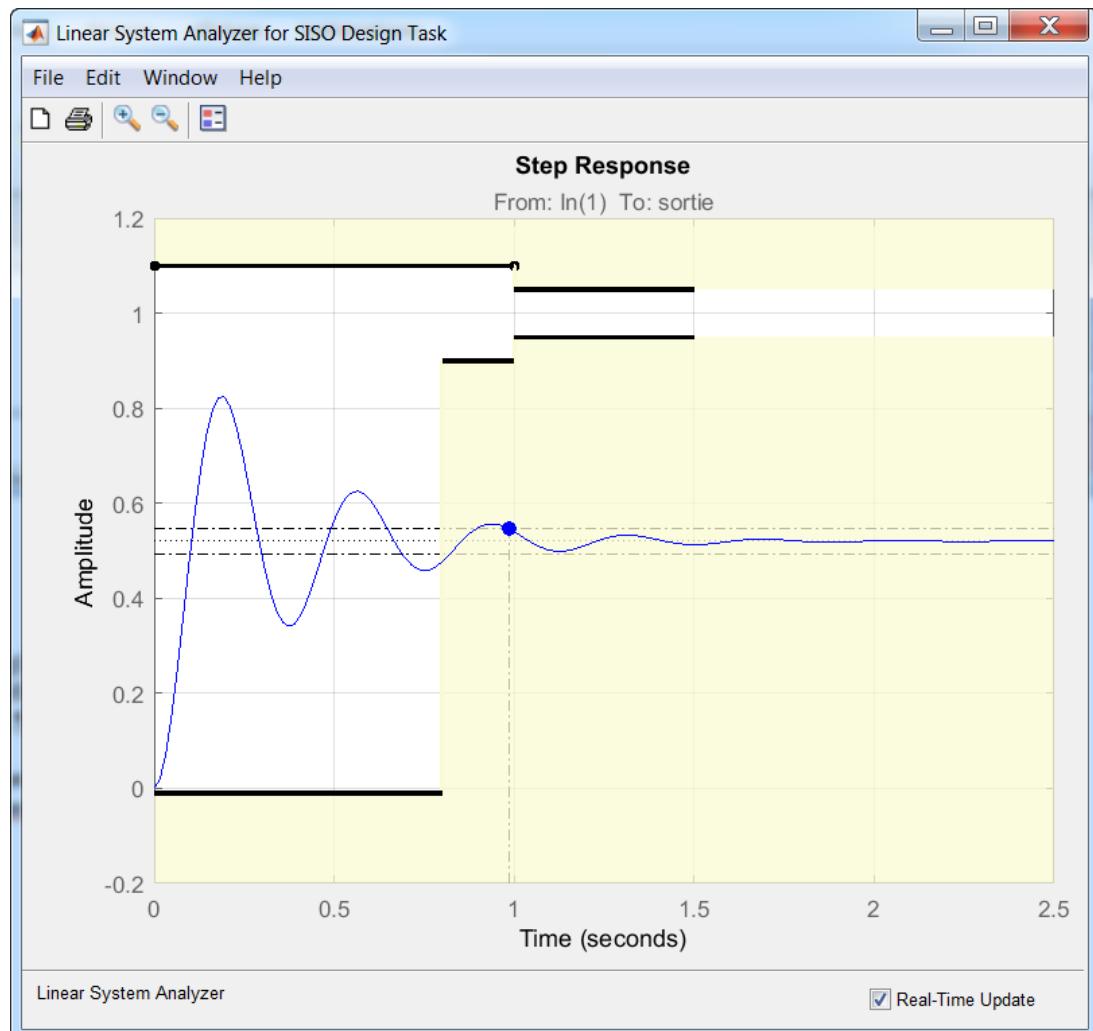


Figure 384: Display of the performance criteria in the graphic window

The step response must be outside of the yellow area, which is defined by the different criteria for the specifications to be respected.

8. Adjusting the PID with the help of cursors

The response is not satisfactory. Manually change the PID settings using the cursors. We can see the plots of the **SISO Design for SISO Design TASK** changing dynamically.
The window in Figure 385 shows a satisfactory PID setting.

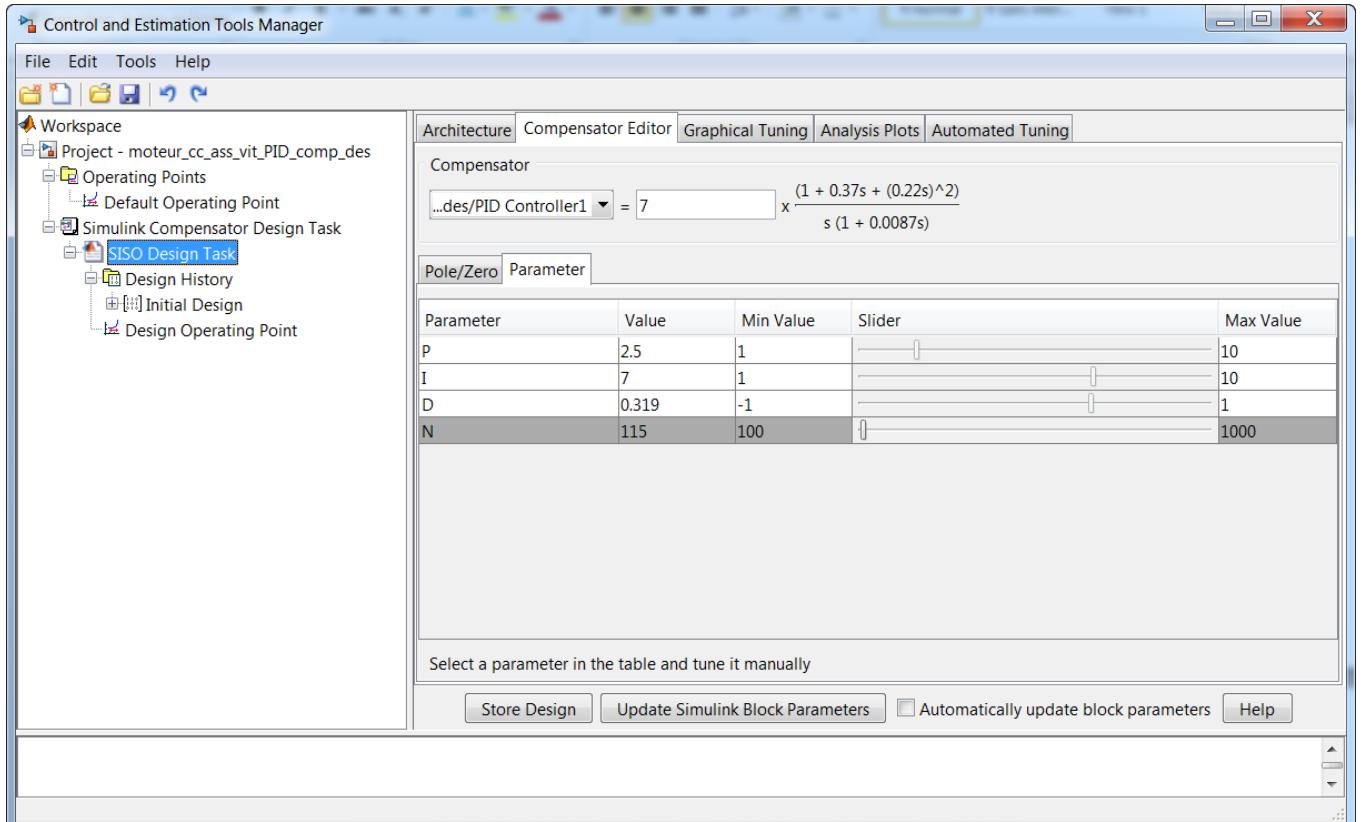


Figure 385: Settings satisfying PID specifications

The window in Figure 386 shows the corrected step response.

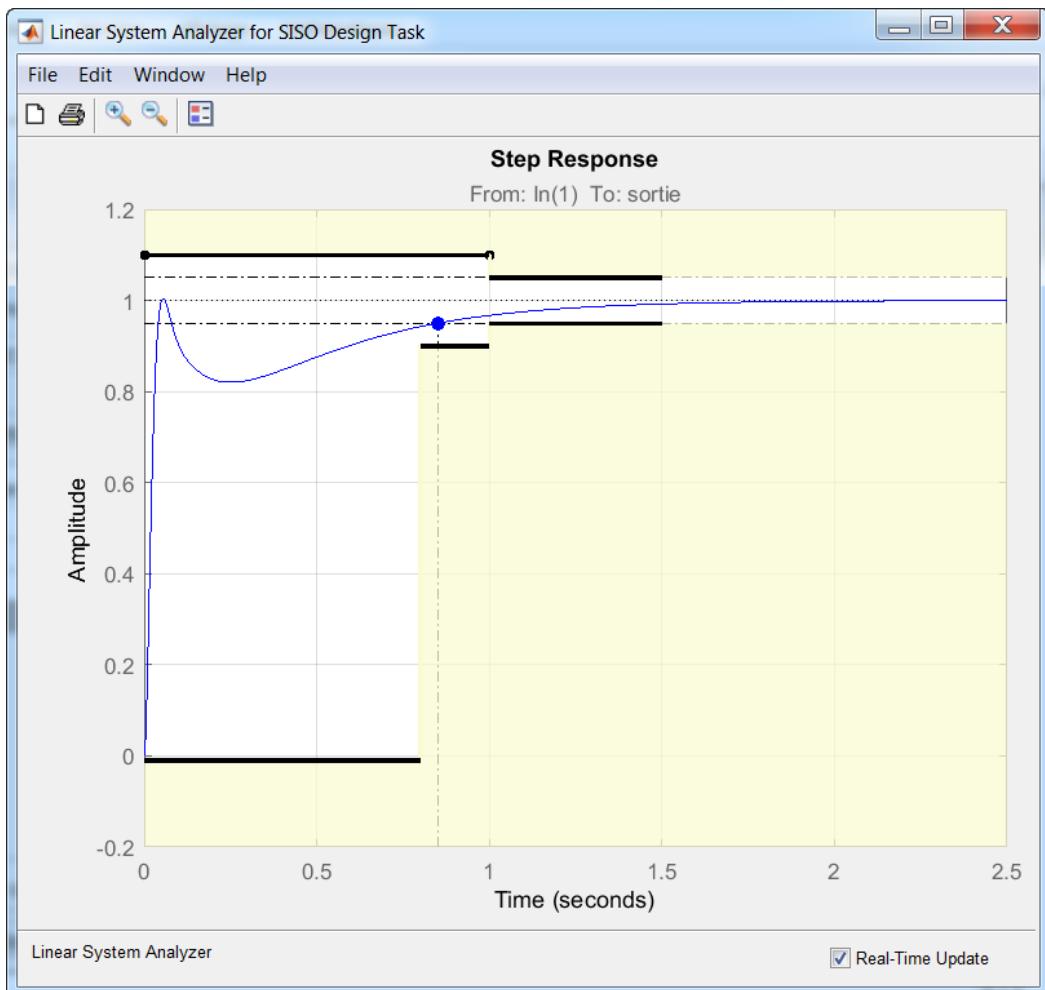


Figure 386 : Results of manual adjustment to the PID on the time domain response

Now that the setting is satisfactory in relation to the guidelines, we can now check the behavior in relation to the disturbance

To do this in the **Control and Estimation Tool Manager** window select the **Analysis Plots** tab then select the step response vis-à-vis disturbance level as indicated in Figure 387.

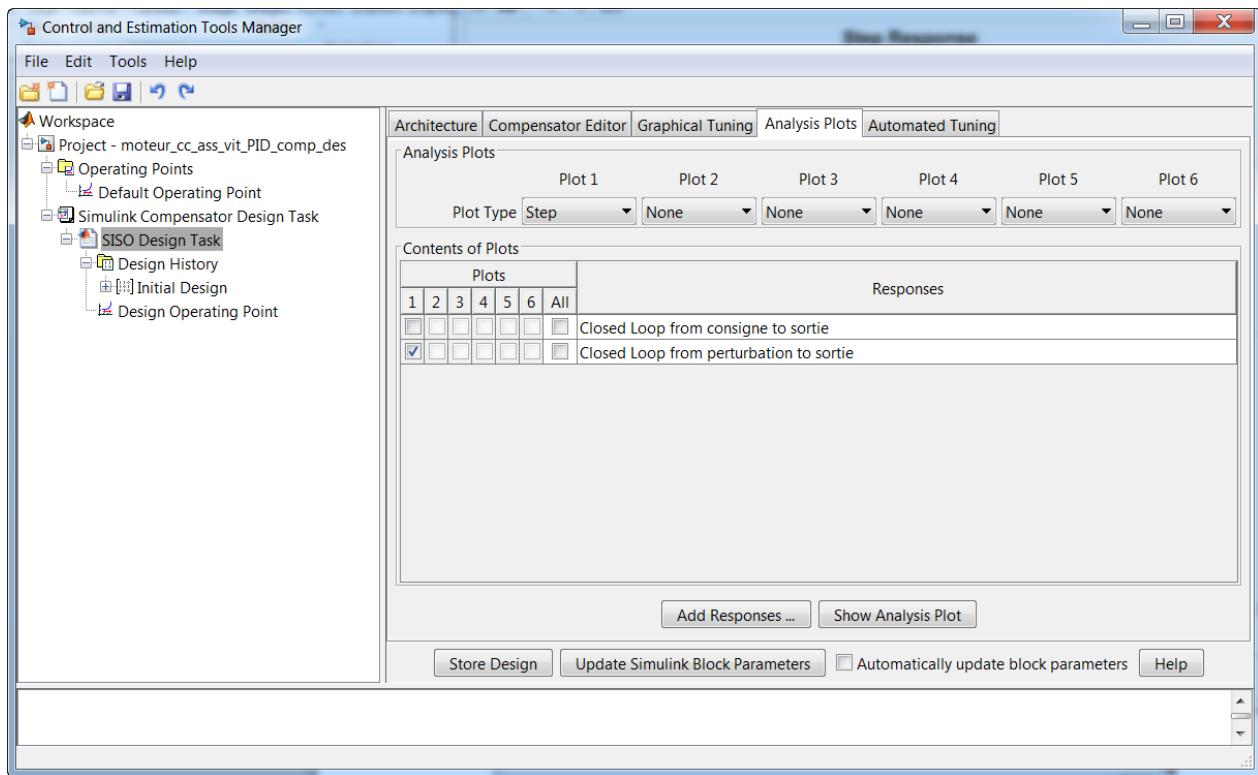


Figure 387: Selecting the step response in accordance with the disturbance

The **Linear Analysis for SISO Design Task** window allows you to view the step response in accordance with the disturbance. The performance criteria of the step response in relation to the guideline should appear again. You can delete them by right-clicking in the yellow part of the graphic window and selecting **Delete**.

The response is then obtained in relation to the disturbance as shown in Figure 388.

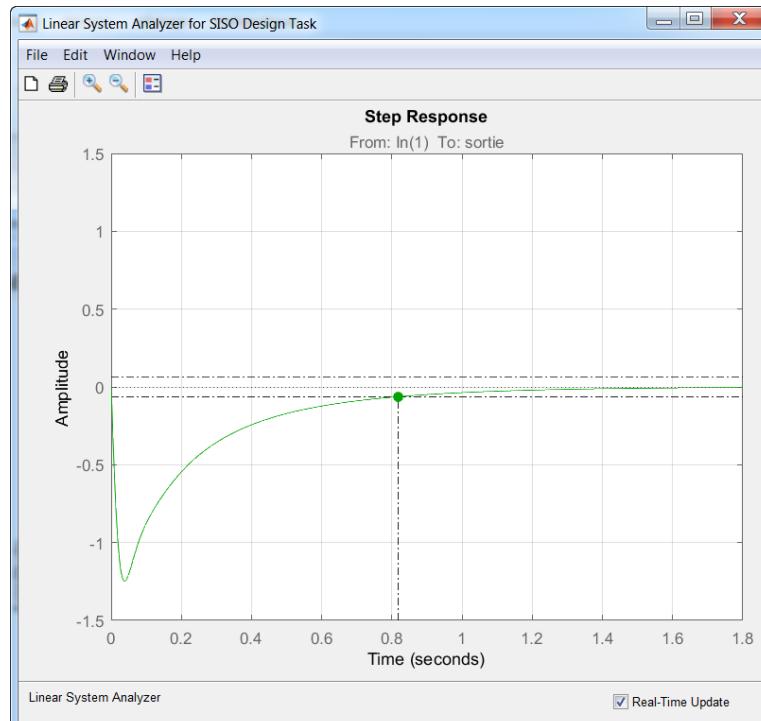


Figure 388: Step response of the system in relation to a level of disturbance

We find that the disturbance is quickly dismissed and that its influence can be considered negligible after 0.8s. The adjustments made are now satisfactory.

The **SISO Design for SISO Design Task** allows you to see the influence of these adjustments on the **Design Plots** and to assess the criteria of the gain margin and phase margin.

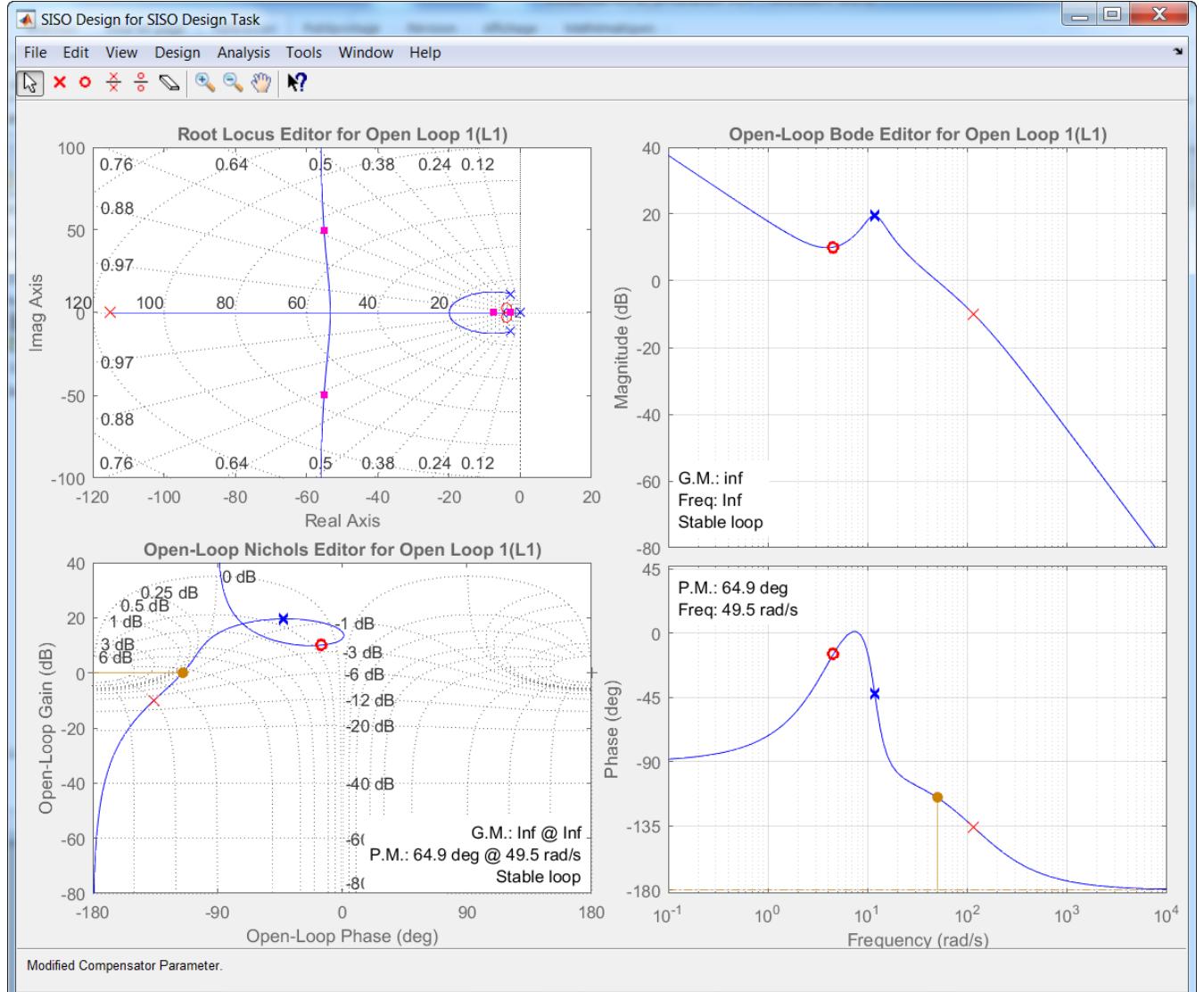


Figure 389: Influence of adjustments on Design Plots

9. Exporting settings in the Simulink model

Now **Click on Update Simulink Block Parameter** in the **Compensator Editor** window to export the PID settings in the PID block of the Simulink model.

A window confirms the importation of the settings parameters in Simulink (Figure 390).

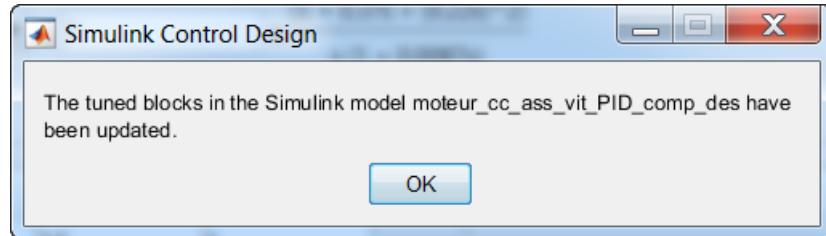


Figure 390: Confirmation of the importation of the parameters of the settings parameters in Simulink

Return to the Simulink model then **Launch** the simulation and view the motor response with the adjusted PID.

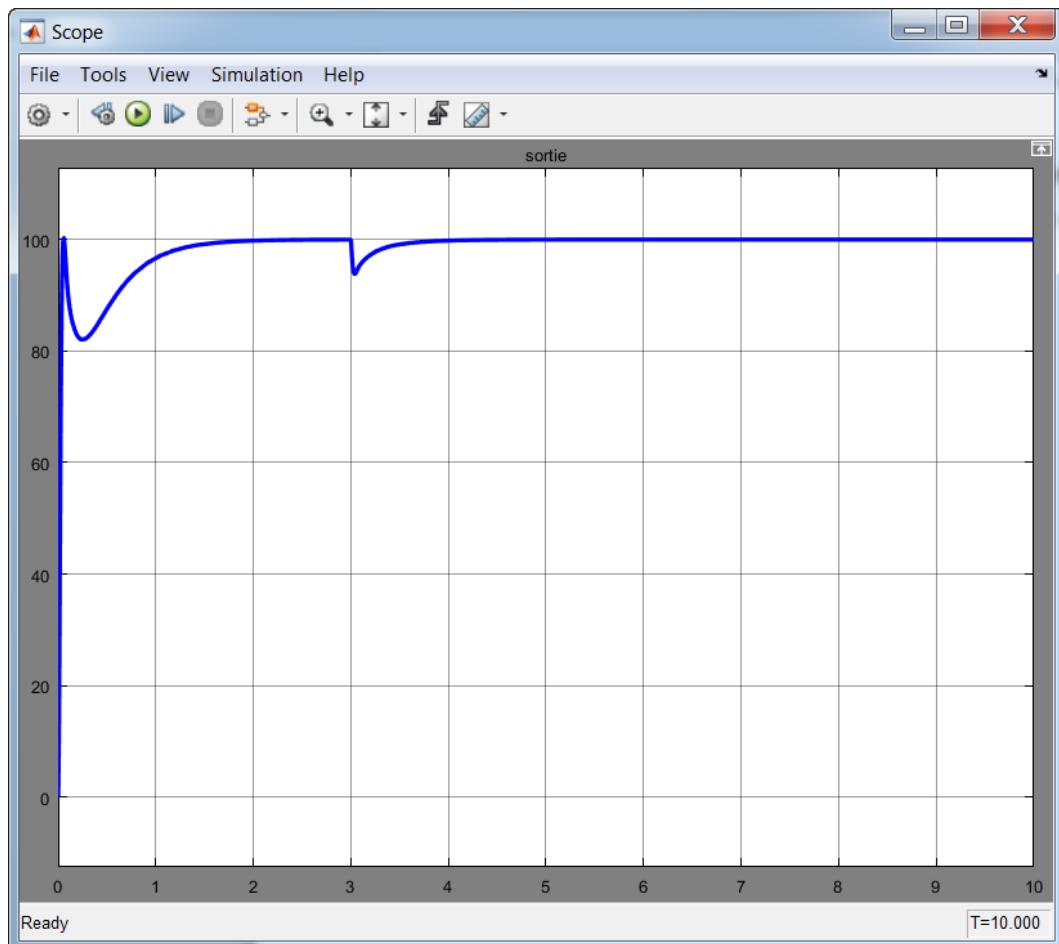


Figure 391: Time domain response after adjusting the PID

We find in Figure 391 that the time domain response is faster, well amortized and that the disturbance is rejected by the system. Performance criteria are respected.

IV. Designing and setting a compensator of any form

In many control applications, the compensator may take forms other than that of a simple PID controller. **MATLAB** provides the opportunity to design any form of compensator by using the “**Control System Designer**” tool. The approach will essentially be the same as that presented in Part III. The transfer function of the compensator will be built step by step by adding new elements in order to improve system performance. The “**Control System Designer**” allows you to integrate into the transfer function of the compensator:

- integrators
- complex poles
- true poles
- complex zeros
- true zeros
- lead compensators (**Lead**)
- lag compensators (**Lag**)
- rejecter filters (**Notch**)

A. Opening the model

Open the file “**moteur_cc_ass_vit_control_sys_des_tf**”

This file contains the control speed model of a DC motor (Figure 392). The compensator is replaced by a transfer function that can be built by using the “**Control System Designer**” tool. For now this transfer function has a pure gain value of 1.

The imposed guideline is 100 rad/s and a disturbance torque is applied to the system at time t=3s.

Linearization points have already been set in this model.

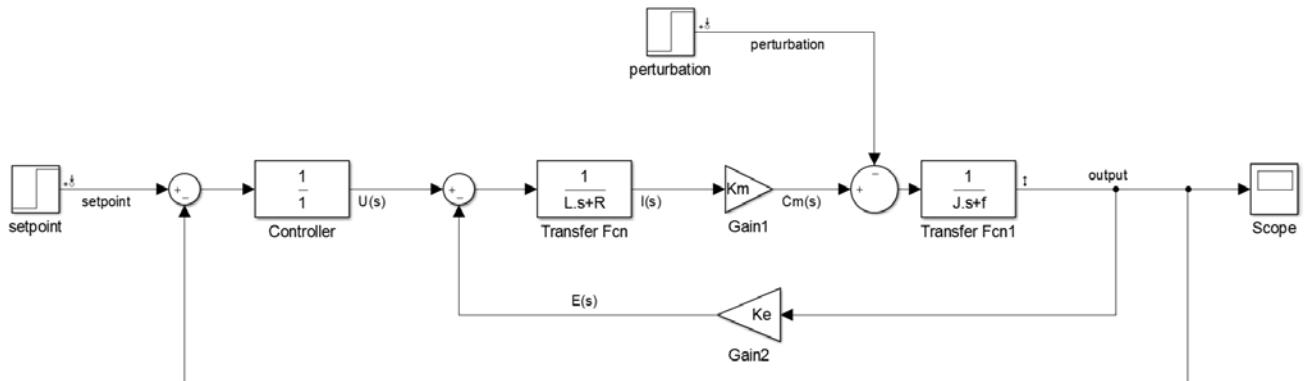


Figure 392: DC motor control model with control by transfer function for adjustments

Run the script *motor_parameters.m*

Launch the simulation and view the response in the scope.

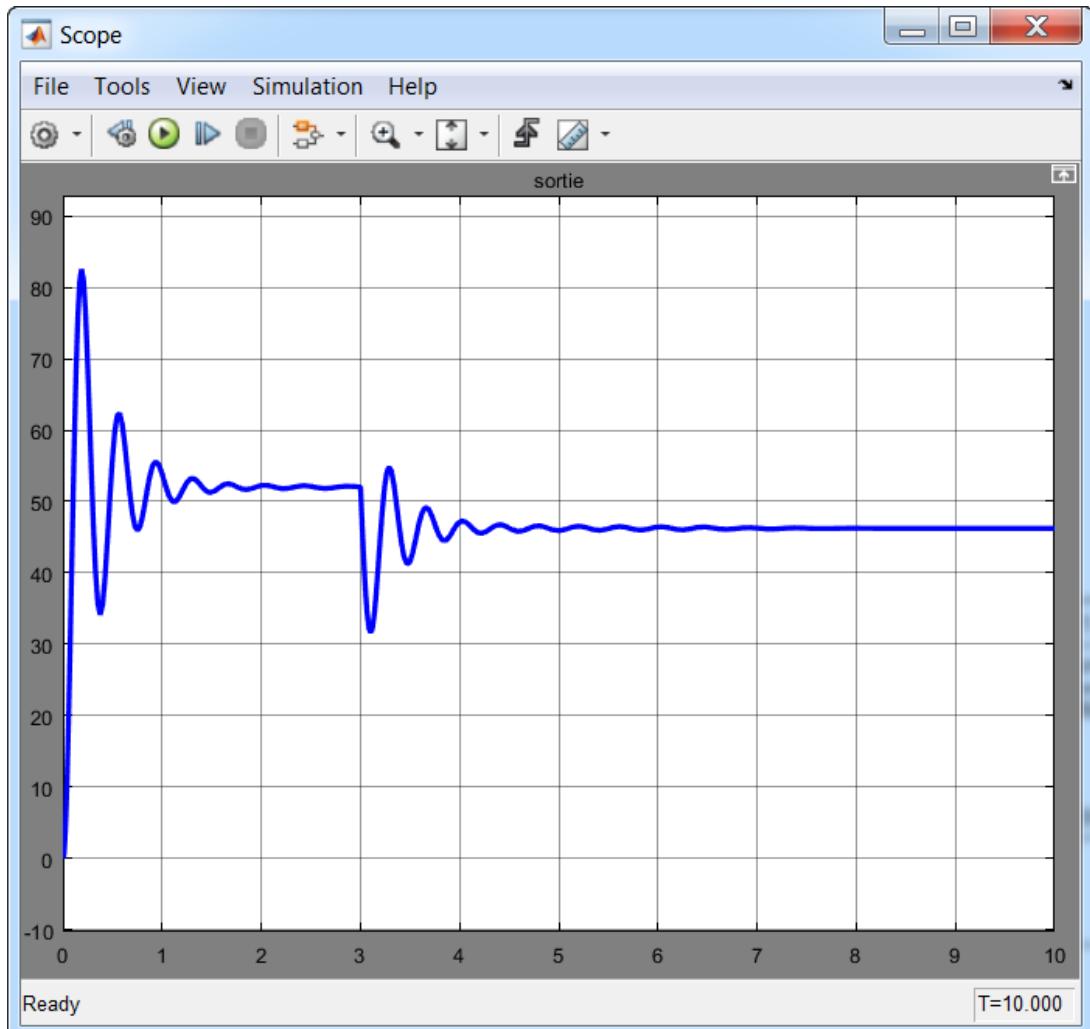


Figure 393: Non-corrected system response

We will find that the system response is not satisfactory. The system is not accurate, poorly damped and does not correctly reject the disturbance.

B. Designing a compensator

From the command bar, select **Analysis/Control Design/Control System Designer**.

This command will open the **Control and Estimation Tools Manager** window that lets you choose the blocks you want to adjust and view the input and output points of the closed-loop.

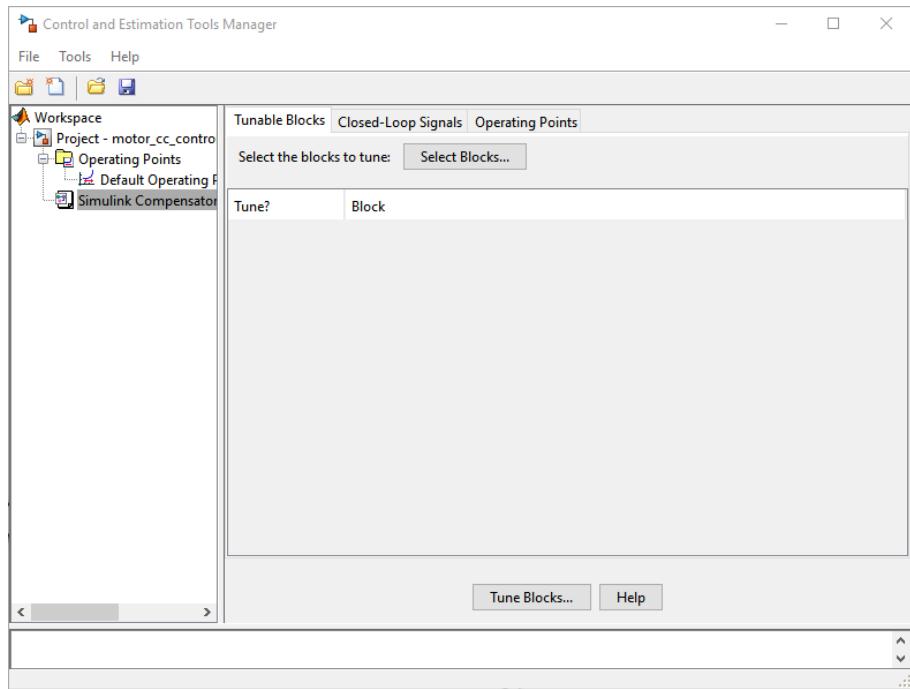


Figure 394: Control and Estimation Tools Manager window

1. Choosing a block to adjust

Select the **Tunable Blocks** tab to choose a block to adjust then click **Select Blocks**.

The **Select Blocks to Tune** window opens and presents all the blocks of the model that can be adjusted.

Select **Controller** (this is the name given in the **Simulink** model to the transfer function of the compensator we wish to design) and click **OK** in order to proceed with the design of the compensator.

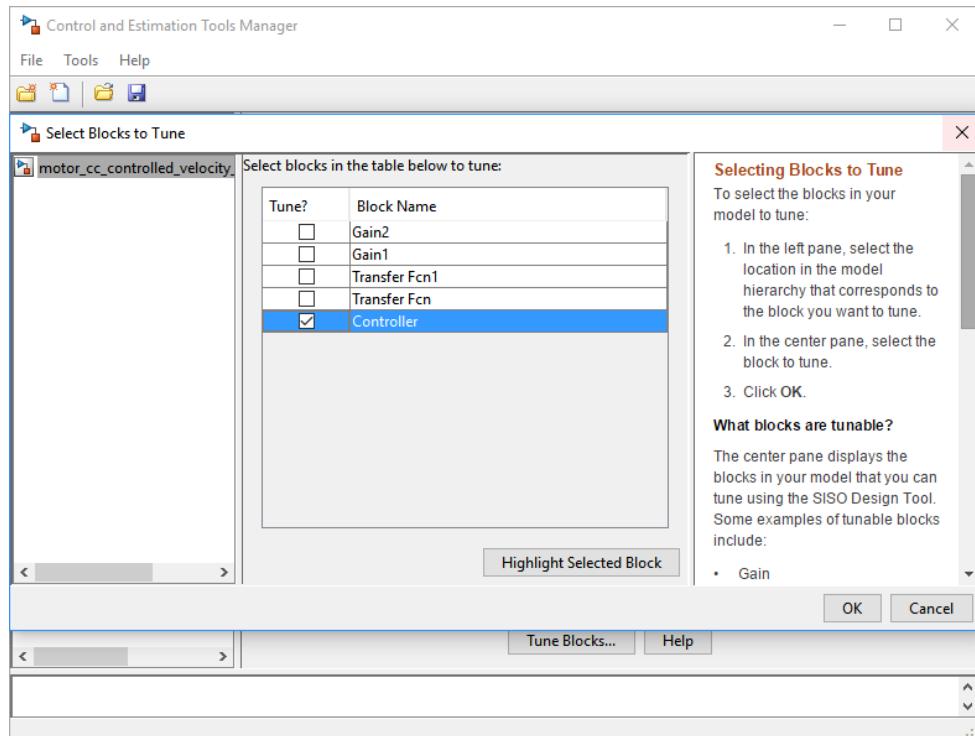


Figure 395: Choosing a block to adjust

Then click on **Tune Block** to begin the adjustments.

An information window (Figure 396), allows you to become familiar with the functionalities of the **Control System Designer** tool.

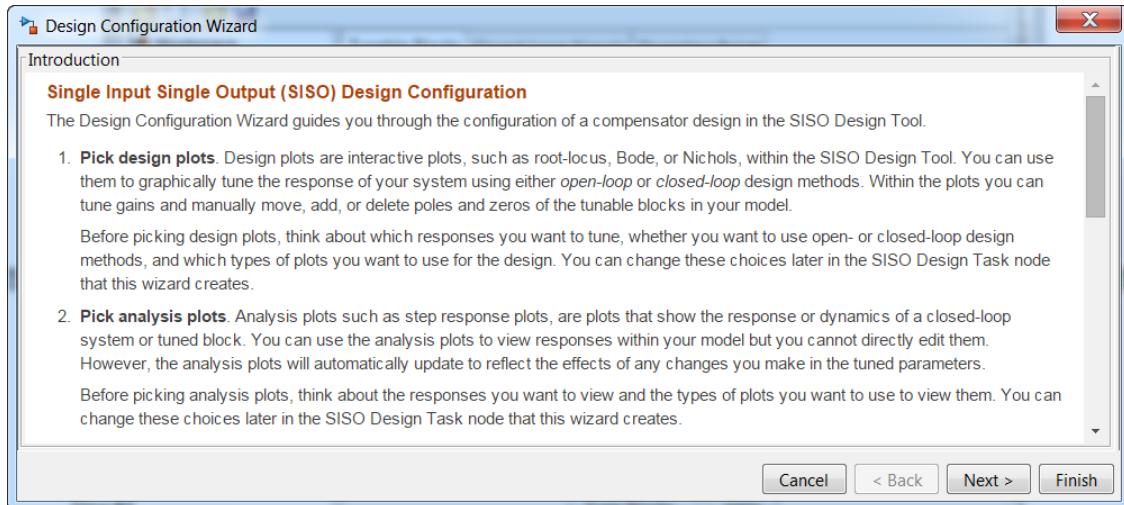


Figure 396: Information window for the Control System Designer tool

Click Next

2. Choosing plots to display for the open-loop

Procedures for choosing plots are described in paragraph III of this chapter.

The **Design Configuration Wizard** window opens and allows you to choose the plots that can be changed to perform the adjustments.

Choose the plots according to Figure 397.

- Root locus (locus of the poles in a **closed-loop transfer function**)
- Bode diagram of the **open-loop transfer function**
- Nichols diagram of the **open-loop transfer function**

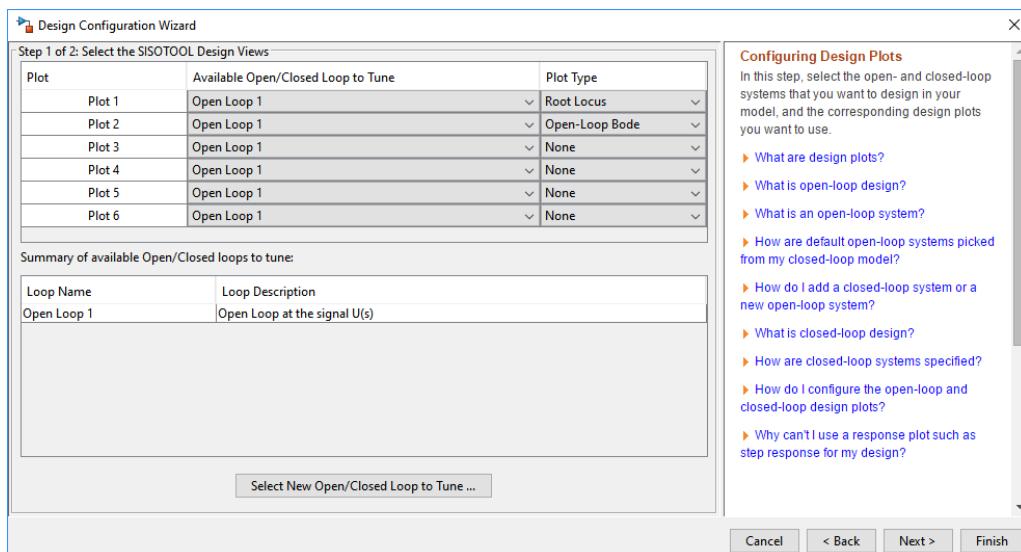


Figure 397: Choosing the poles to view for the open-loop

3. Choosing plots to view the performance of the closed-loop

To view the closed-loop transfer function performance, we can display:

- The time domain response during a step of the **closed-loop transfer function**
- The Bode diagram of the **closed-loop transfer function**

Here we are only interested in the behavior of the loop system in relation to the input.

Configure the Design Configuration Wizard window according to Figure 398.

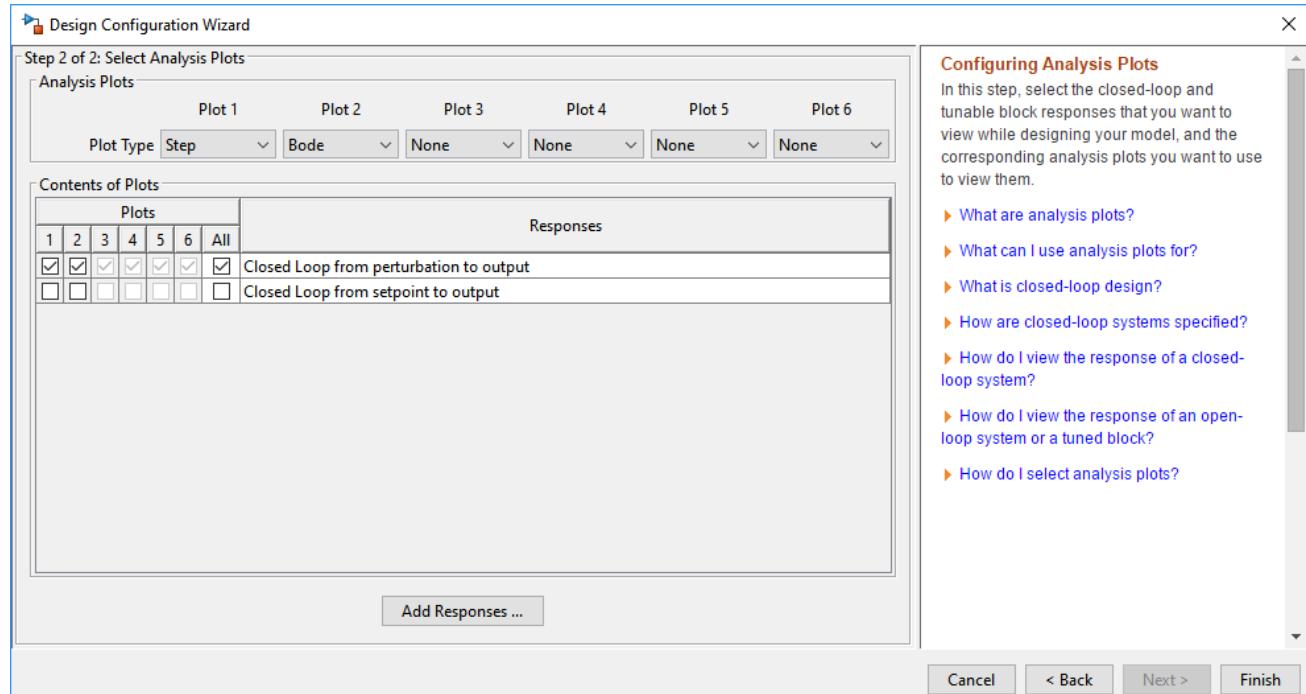


Figure 398: Selecting display plots of the closed-loop performance

Click Finish

4. Analyze the graphic windows of the “compensator design” tool

The two windows **Linear System Analyzer for SISO Design Task** (allows the display of closed-loop transfer function performance) and **SISO Design for SISO Design Task** (allows the display of the behavior of the open-loop transfer function)

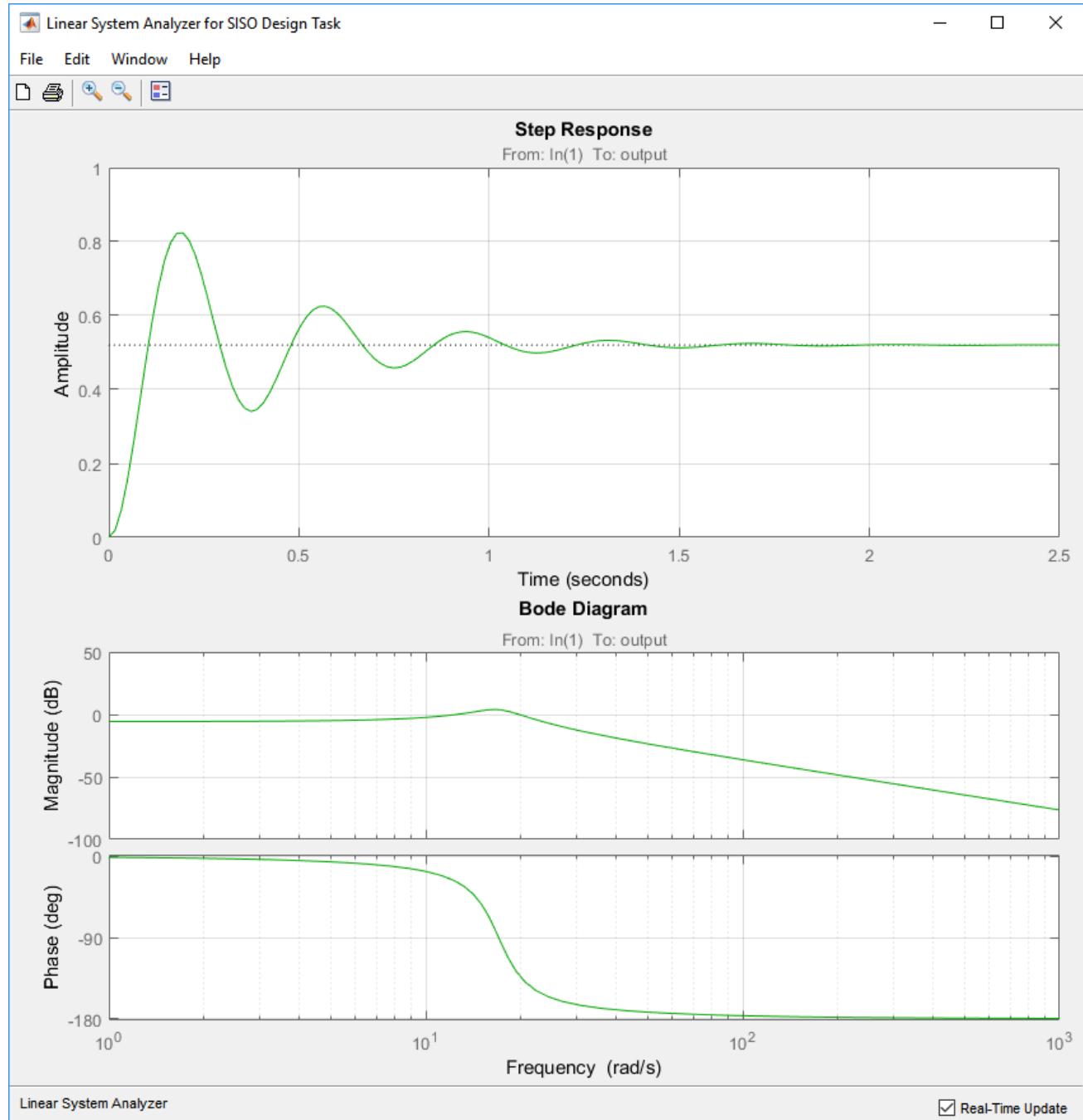


Figure 399: Display of closed-loop transfer function performance

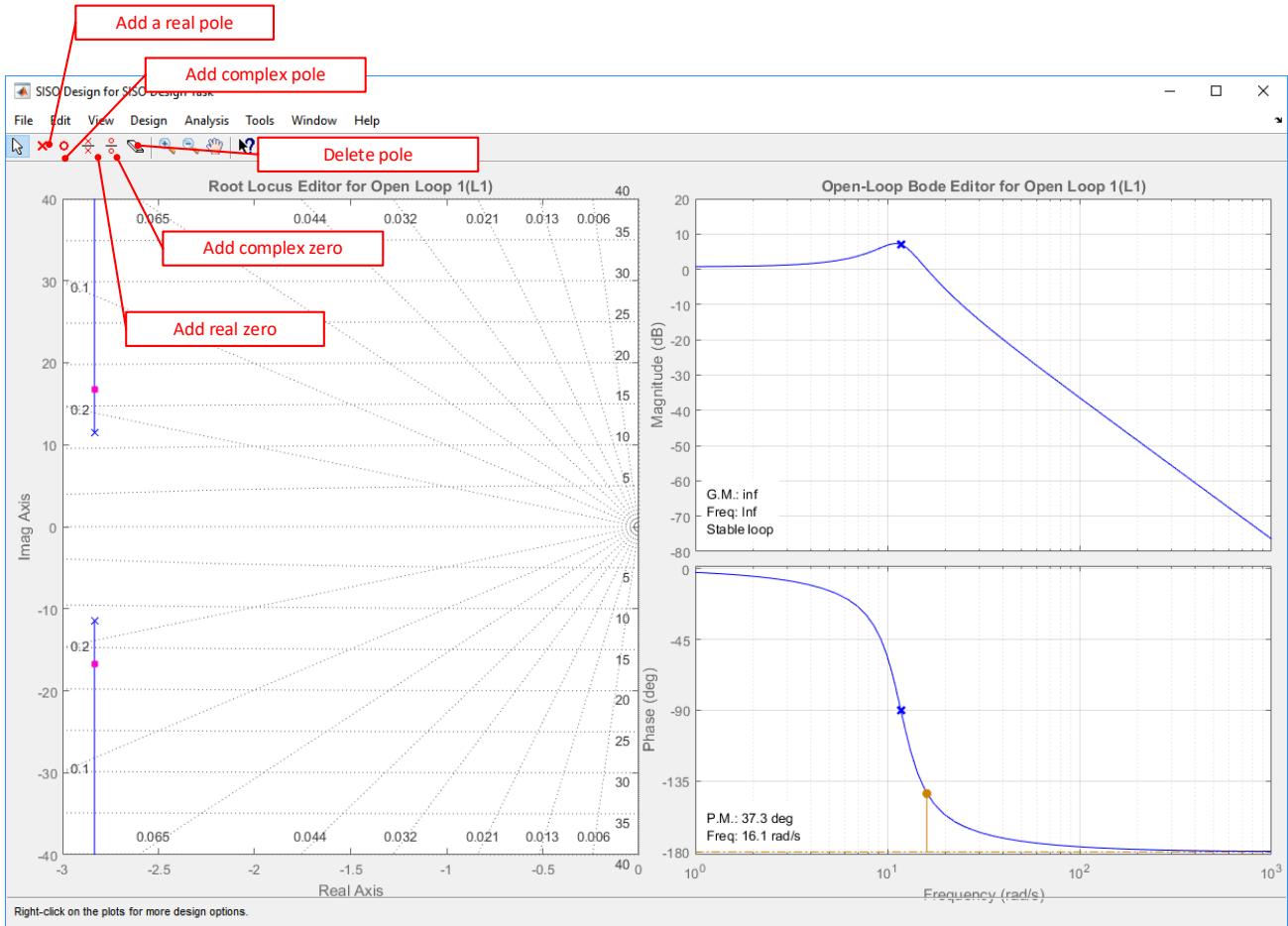


Figure 400: The command bar of the SISO Design for SISO Design Task window

Note that the **Control and Estimation Tool Manager** window also stays open. The **Compensator Editor** tab is very important since it will allow you to visualize the shape of the transfer function that we will build.

To use the tool, it is best to view these three windows simultaneously:

- **Control and estimation Tool Manager** to make adjustments
- **SISO Design for SISO Design Task** to see the influence of the adjustments on the behavior of the open-loop
- **Linear System Analyzer for SISO Design Task** to see the influence of the adjustments on the behavior of the closed-loop

5. Viewing the gain influence of the closed-loop transfer function

It is possible to initially vary the gain of the compensator and to view its influence on all diagrams.

In the **SISO Design for SISO Design Task** window, on the Bode diagram in the open-loop transfer function gain, **move** the mouse pointer over the gain curve. When it takes the form of a hand, use **drag and drop** to pan the curve vertically.

This translation will have the influence of changing the gain value of the compensator that we are designing.

Move the gain curve to obtain a low frequency gain of about **20dB** for the open-loop transfer function.

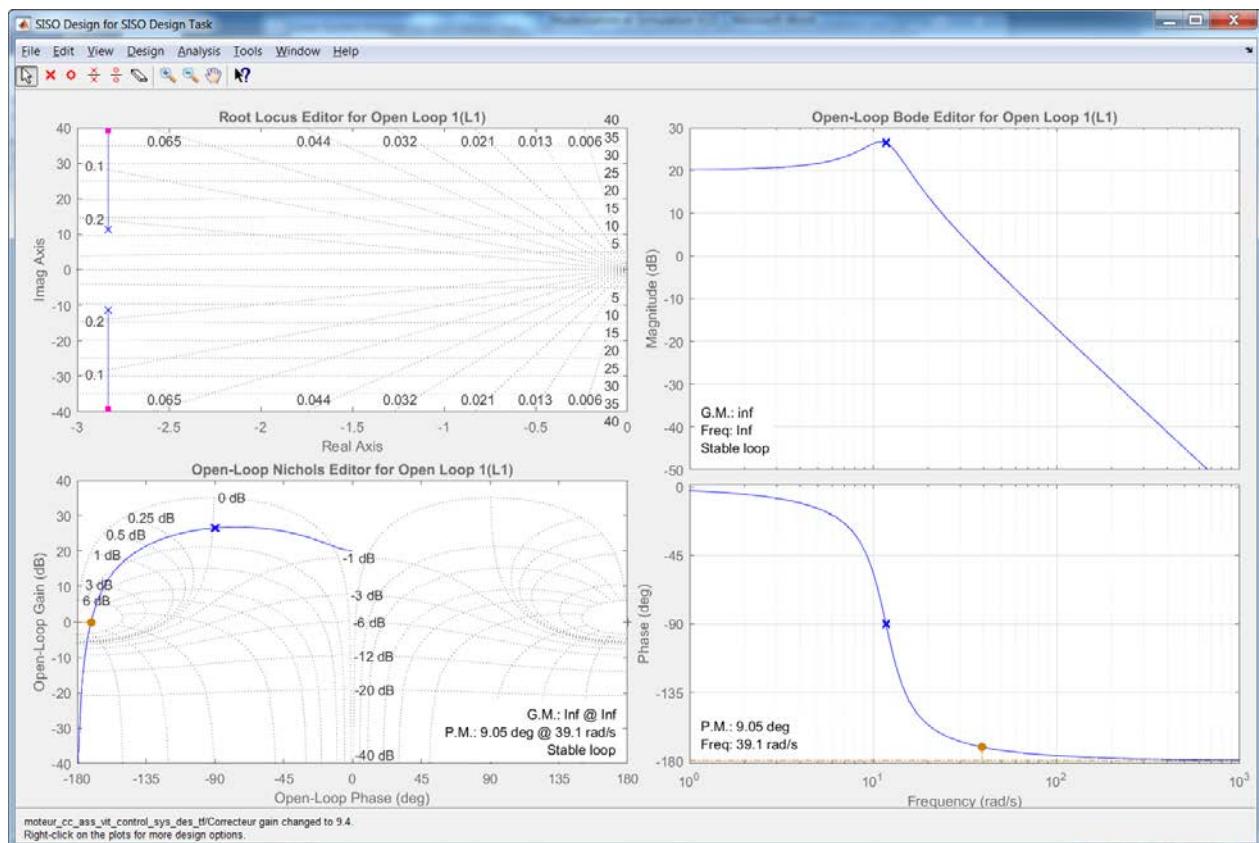


Figure 401: Gain variation of the open-loop transfer function

We find that the other diagrams are updated dynamically by taking into account the increase in the gain of the open-loop transfer function.

The influence of increasing the proportional gain of the compensator on the behavior of the closed-loop is visible in the **Linear System Analyzer for SISO Design Task** window. On the Step response, the accuracy and speed increase, but the depreciation decreases, revealing stronger oscillations. On the Bode diagram, an increase of the bandwidth at -3dB is observed (improving speed) and a higher resonance.

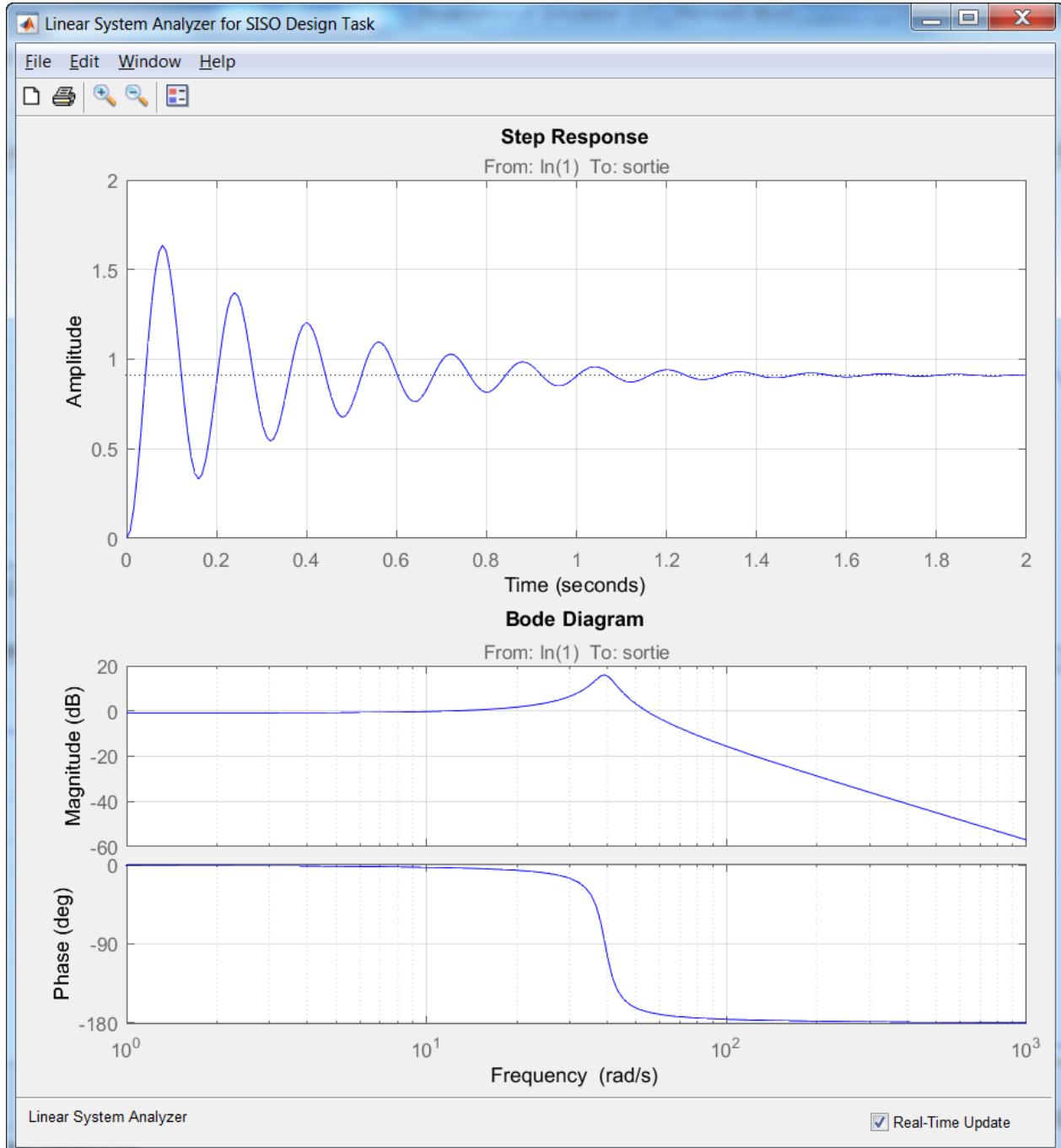


Figure 402: View of the influence on the closed-loop performance

It is also possible to view the new compensator function in the **Control and Estimation Tool Manager** window, **Compensator Design** tab. We can see a pure gain of about 10.

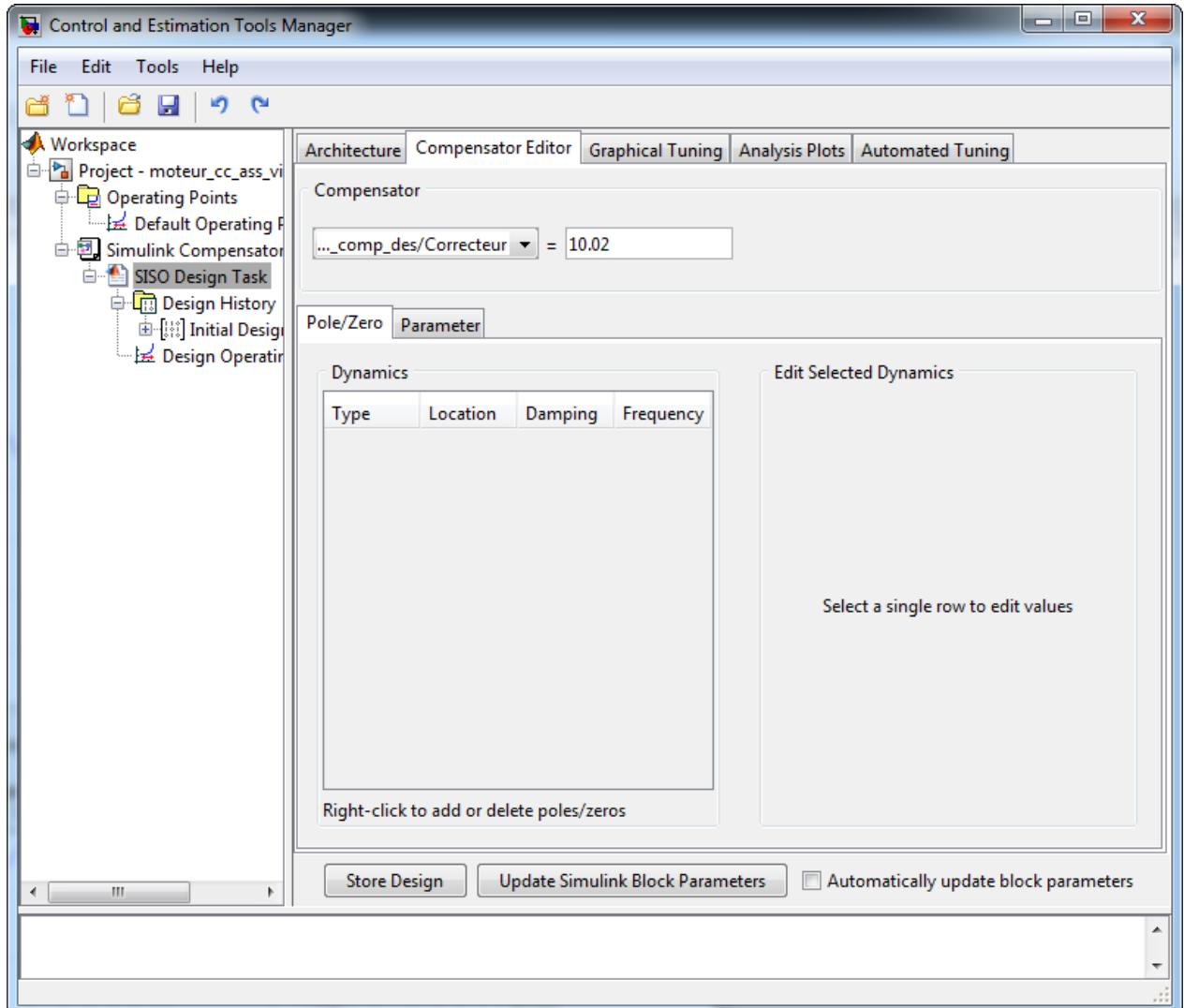


Figure 403: Displaying the compensator transfer function

It is also possible to modify the compensator gain by using other procedures:

- By moving the poles of the FTBF (closed-loop transfer function) along the locus of the poles (root locus)
- By vertically translating the Nichols plot
- By directly indicating the gain value in the **Compensator Design** tab of the **Control and Estimation Tool Manager** window.

In all cases the other diagrams update automatically.

6. Adding an integrator

To make the system accurate, it is possible to add an integrator into the compensator.

To do this, Right-Click the mouse in the **SISO Design for SISO design Task** window then select **Add Pole zero/Integrator**. An integrator is added to the compensator and all curves are updated.

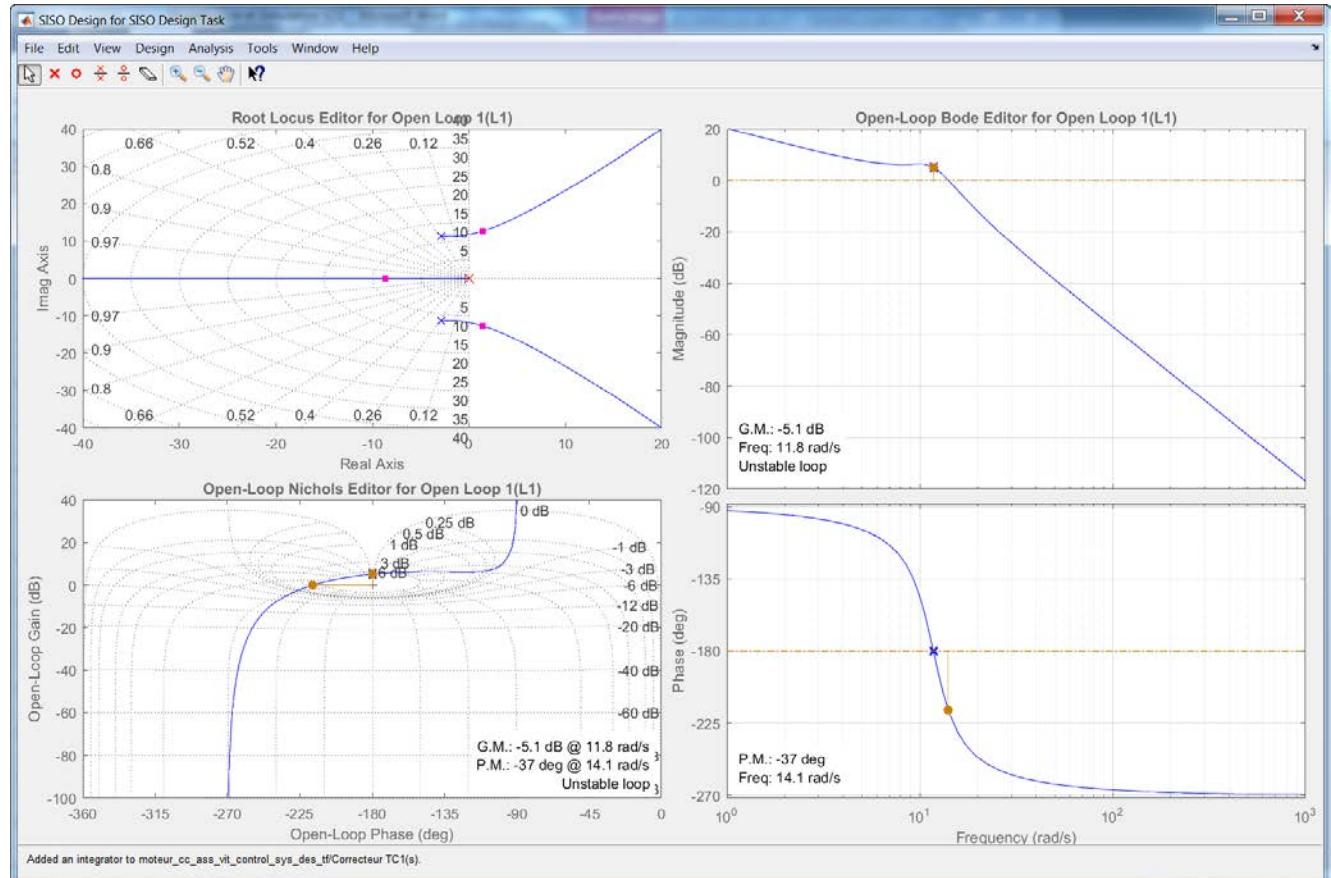


Figure 404: Adding an integrator

Note on the locus of the poles, that the poles of the closed-loop transfer function are actually a positive part of what makes the system unstable. Gain and phase margins are negative and are visible on the Bode diagram and the Nichols plot.

The step response shows this instability (Figure 405).

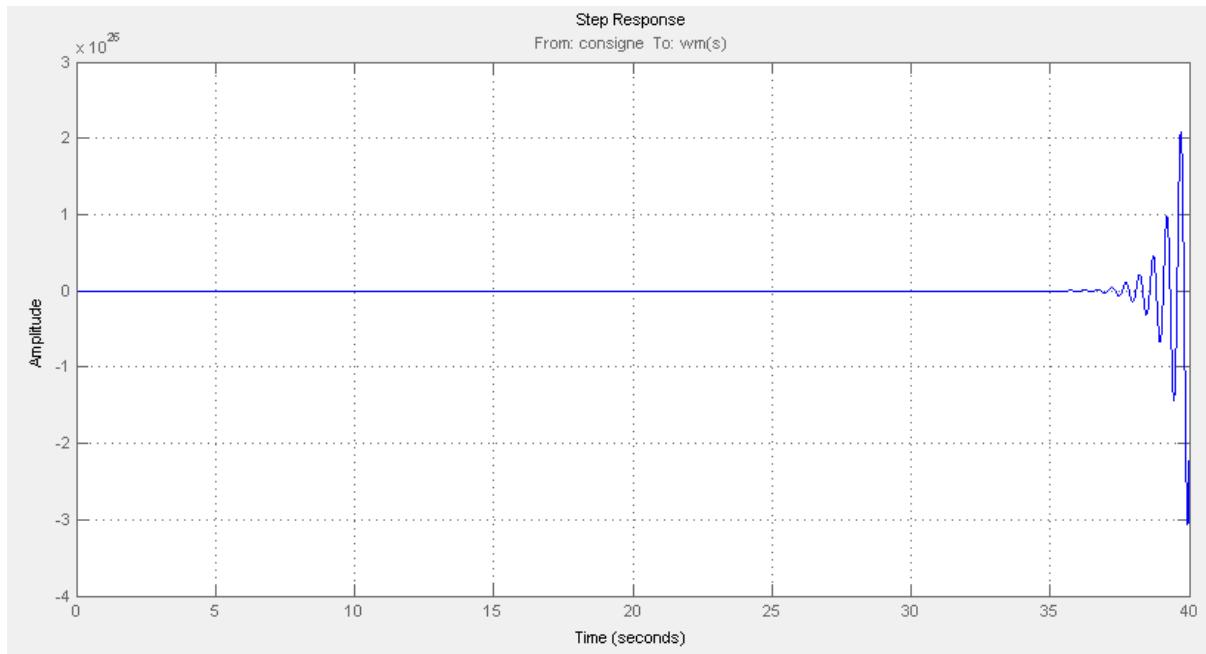


Figure 405: Visualization of the instability of the step response

7. Adding a lead compensator (Lead)

To increase gain and phase margins, it is possible to add a lead compensator (Lead).

To do this **right-click** the mouse, then select **Add Pole zero/Lead**.

Using the mouse pointer, click on the cut-off frequency on the Bode diagram of the open-loop transfer function gain. The pole of the lead compensator will then be set at the cut-off frequency. The zero will automatically be set to a lesser frequency.

It is possible to view the poles and zeros on all curves (Figure 406).

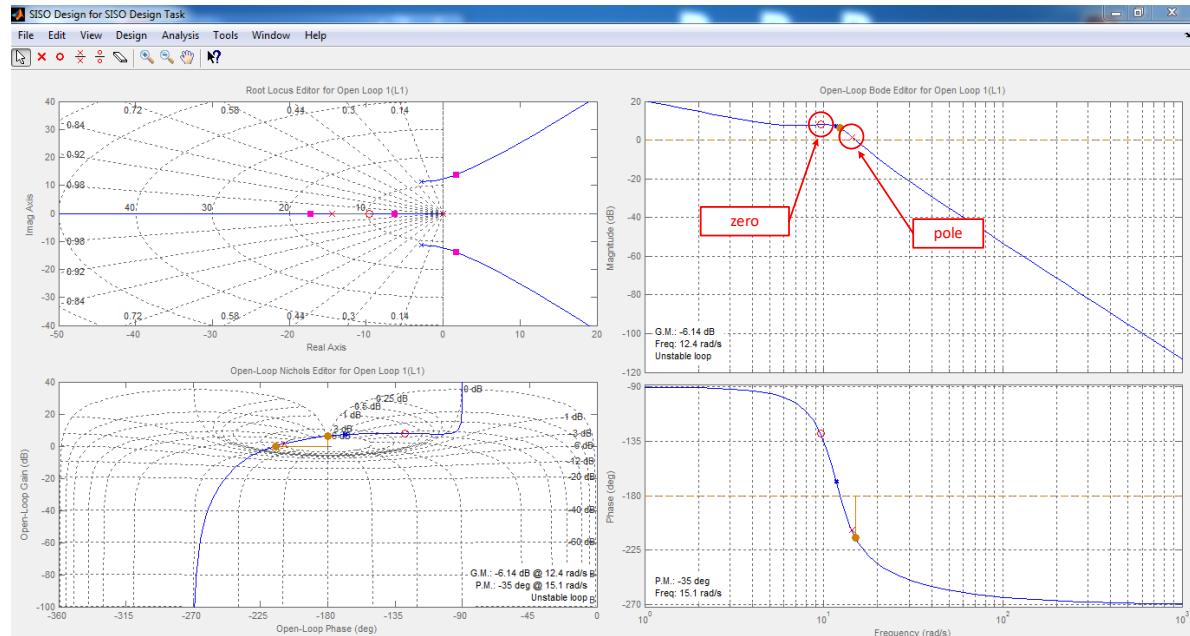


Figure 406: Visualization of the poles and zeros on the curves

To adjust the lead compensator, just use the mouse pointer to move the already added pole and zero to obtain a positive gain and phase margin in order to make the system stable.

Position the **pole** around 1000 rad/s pulsation and the **zero** around 4 rad/s pulsation.

The gain and phase margins are now positive and the system is stable. The poles of the closed-loop transfer function are actually a negative part (Figure 407).

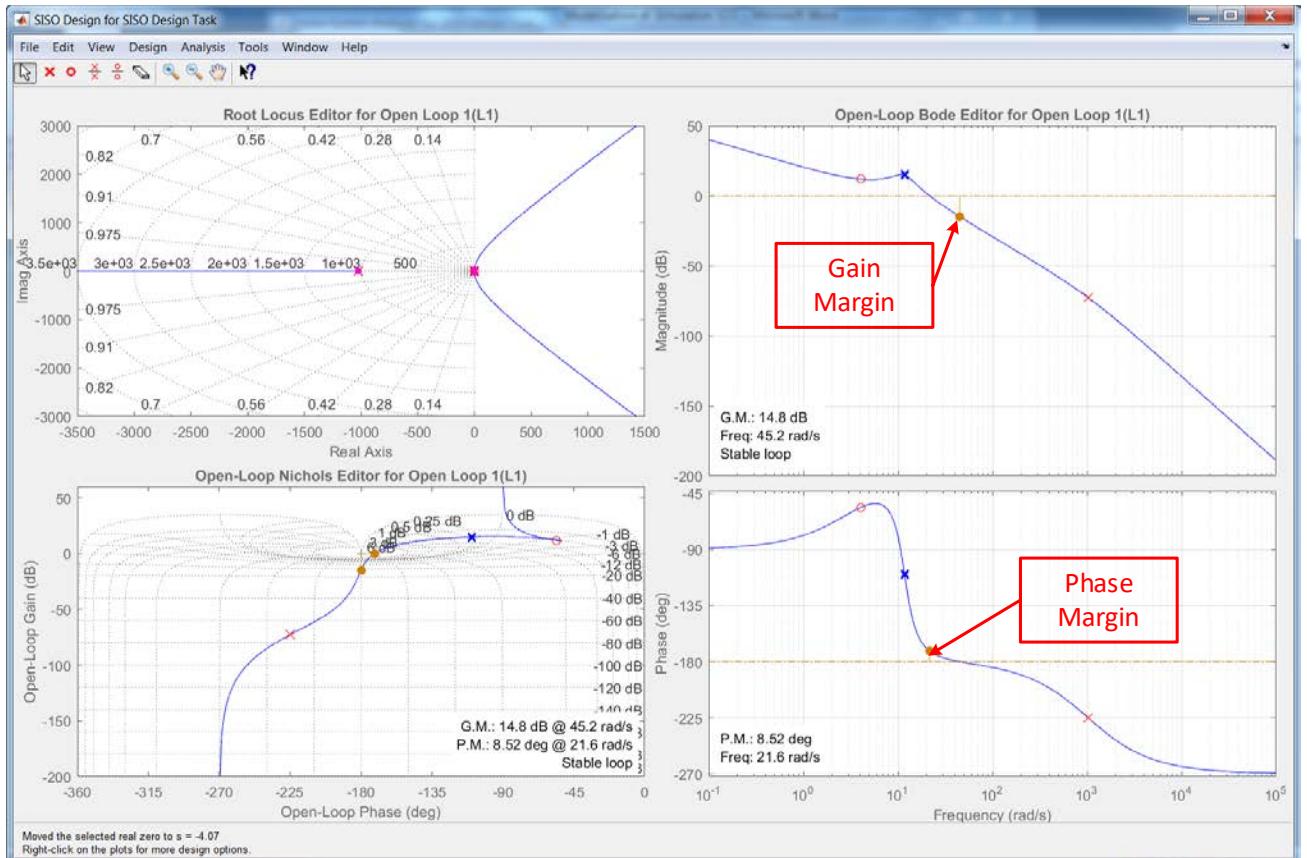


Figure 407: Stabilizing the system by adding a lead compensator

The step response converges and accuracy is good.

The Bode diagram gain of the closed-loop transfer function shows a strong resonance around the pulsation 22 rad/s, which reflects the poor quality of the step response depreciation (Figure 408).

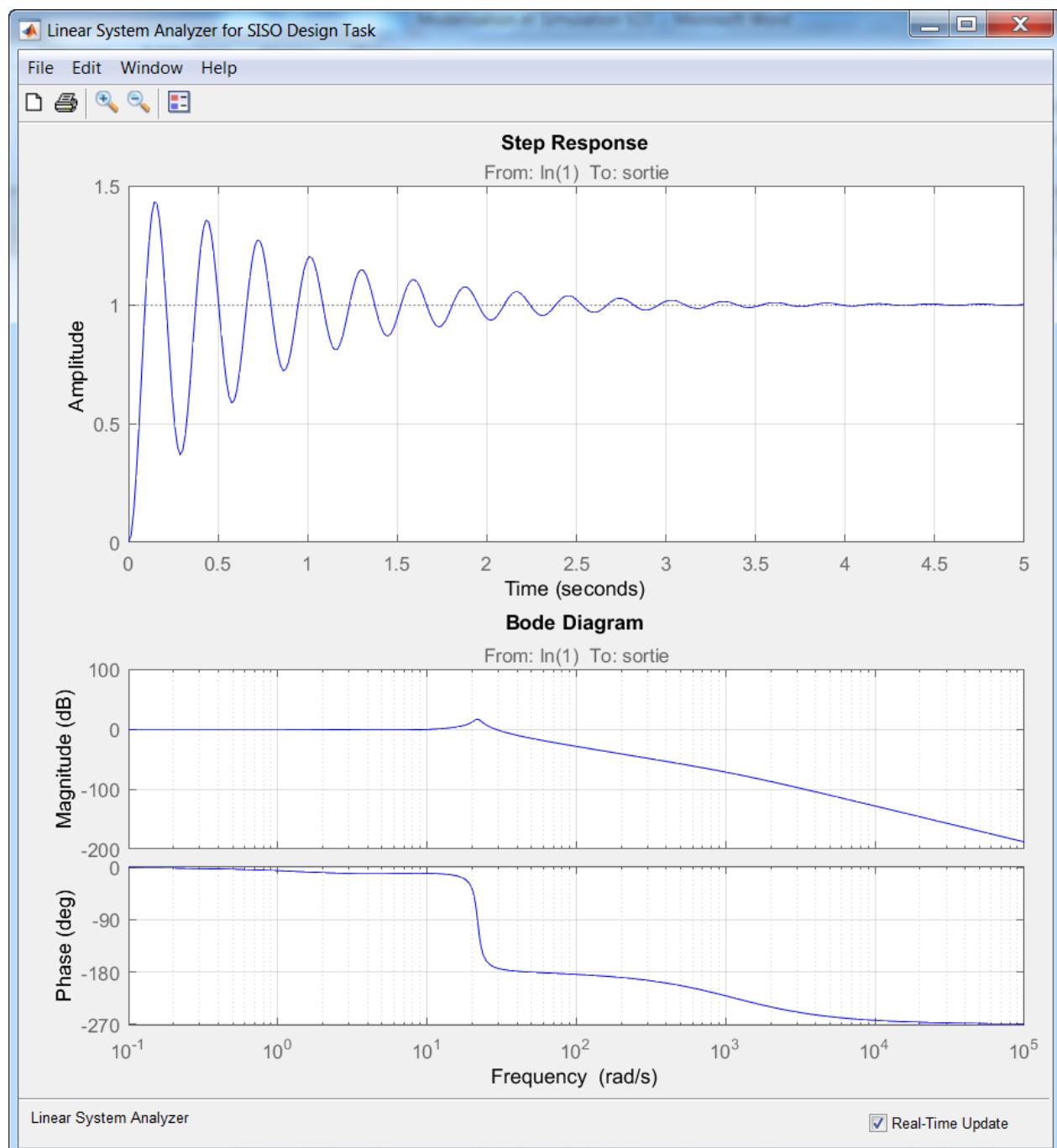


Figure 408: Closed-loop performance after lead compensation

At this stage it is possible to see the compensator transfer function in the **Control and Estimation Tool Manager** window in the **Compensator Design** tab (Figure 409).

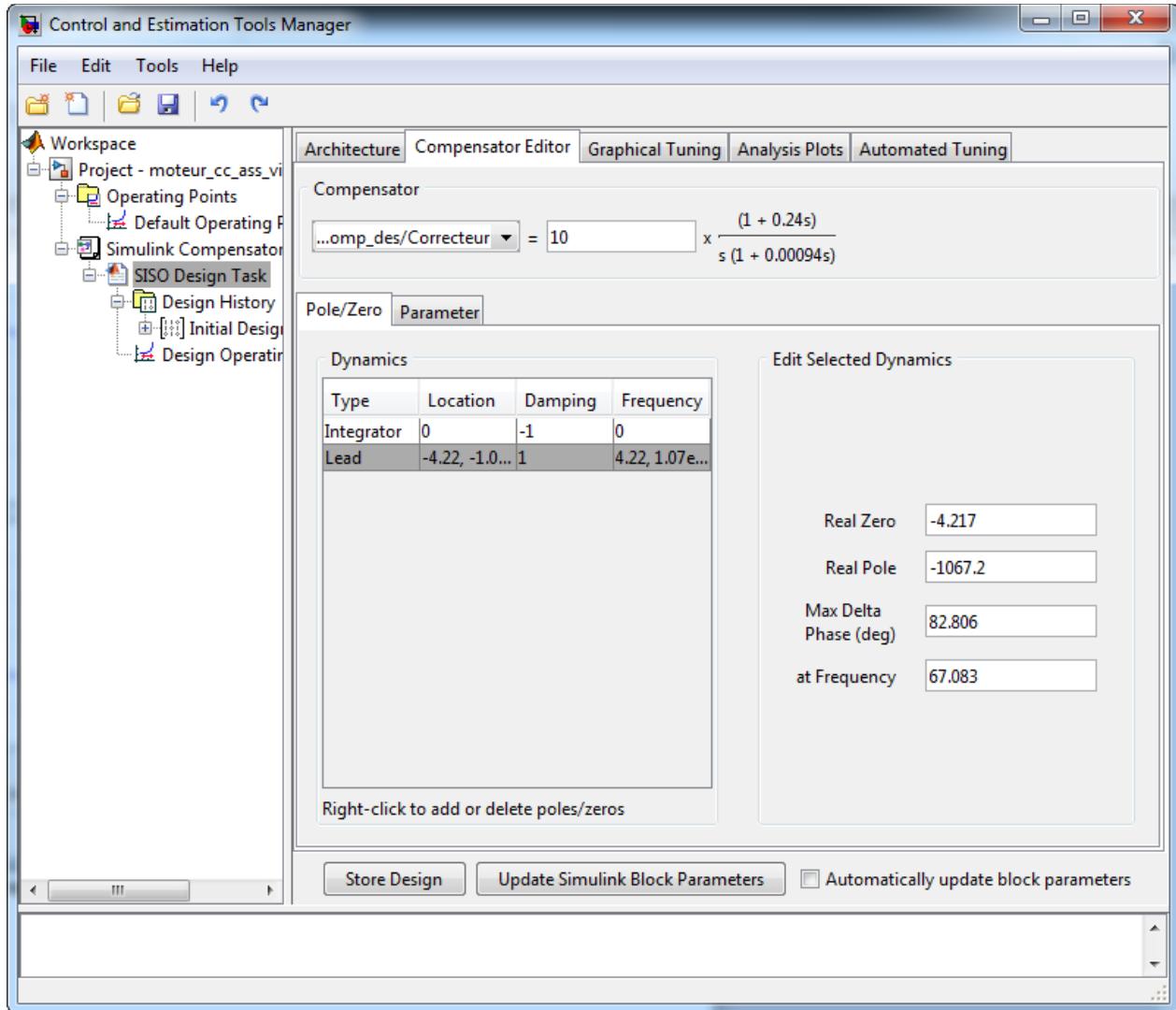


Figure 409: Viewing the compensator transfer function

8. Adding a Notch

Compensating for the resonance phenomenon found in the Bode diagram of the closed-loop transfer function (peak at a pulsation of 20 rad/s). It is possible to add a Notch. This filter will reduce resonance and reduce oscillations.

To do this, **right-click** the mouse, then select **Add Pole zero/Notch**.

Then click on the Bode diagram gain of the open-loop transfer function slightly off from the resonance peak observed on the Bode diagram of the closed-loop transfer function (about 10 rad/s).
The rejecter filter is now added to the compensator and the diagrams are updated (Figure 410).

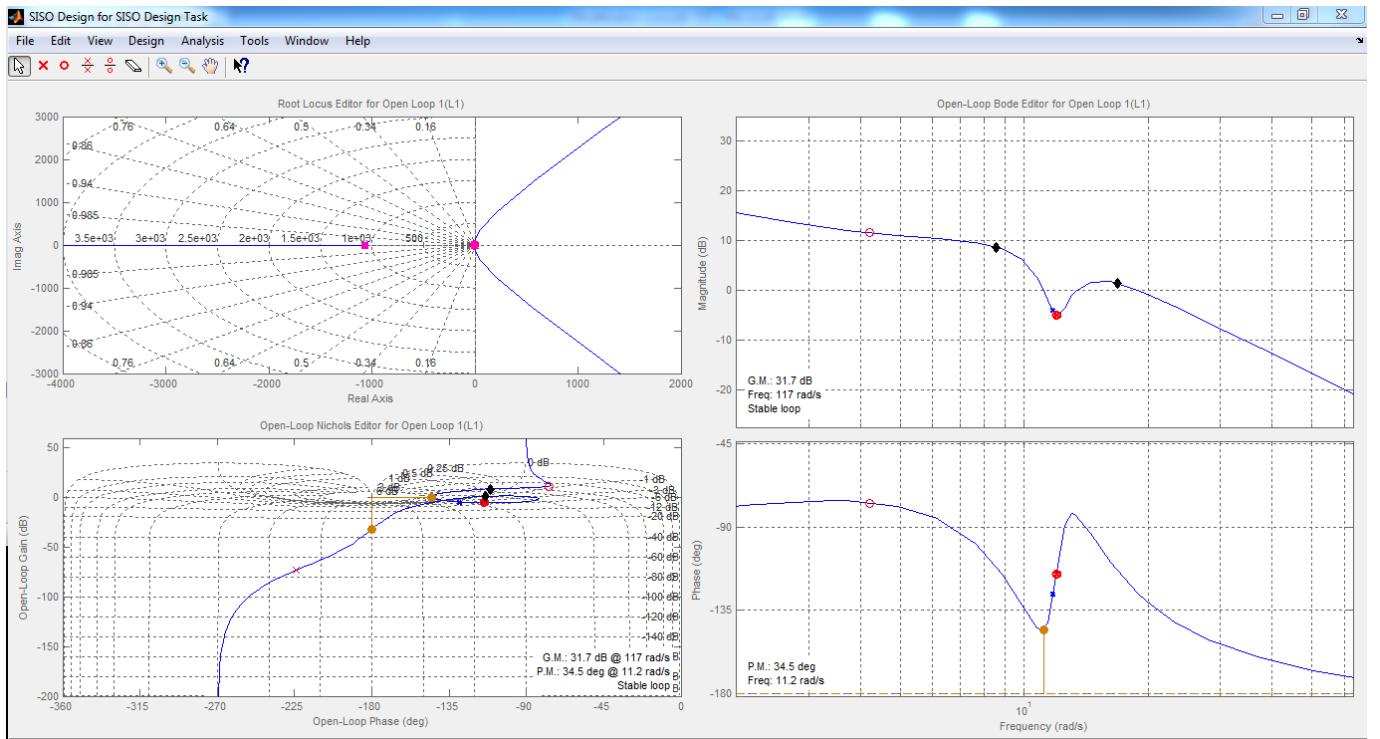


Figure 410: Adding a rejecter filter to the compensator

In Figure 411 we can see the influence on the step response and on the Bode diagram of the closed-loop. It appears that the filter is not adjusted to the extent that the resonance of the Bode diagram of the closed-loop is not completely compensated and major fluctuations persist.

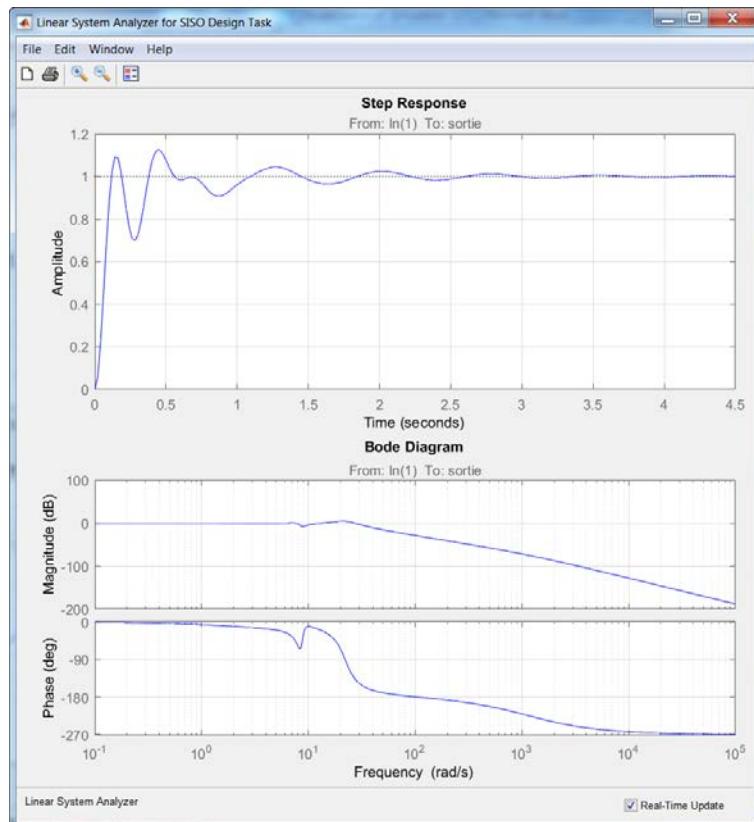


Figure 411: Influence of the notch on the performance of the closed-loop

9. Adjusting a Notch

To adjust the Notch, it is possible to graphically modify 3 parameters:

- The frequency at which the filter will operate (**Natural Frequency**)
- Depth in dB that is desired (**Notch Depth**)
- The frequency range in which the filter must operate (**Notch Width**)

All these parameters can be adjusted through the gain Bode diagram of the open-loop transfer function by simply dragging and dropping the heights representing the parameters of the filter indicated in Figure 412.

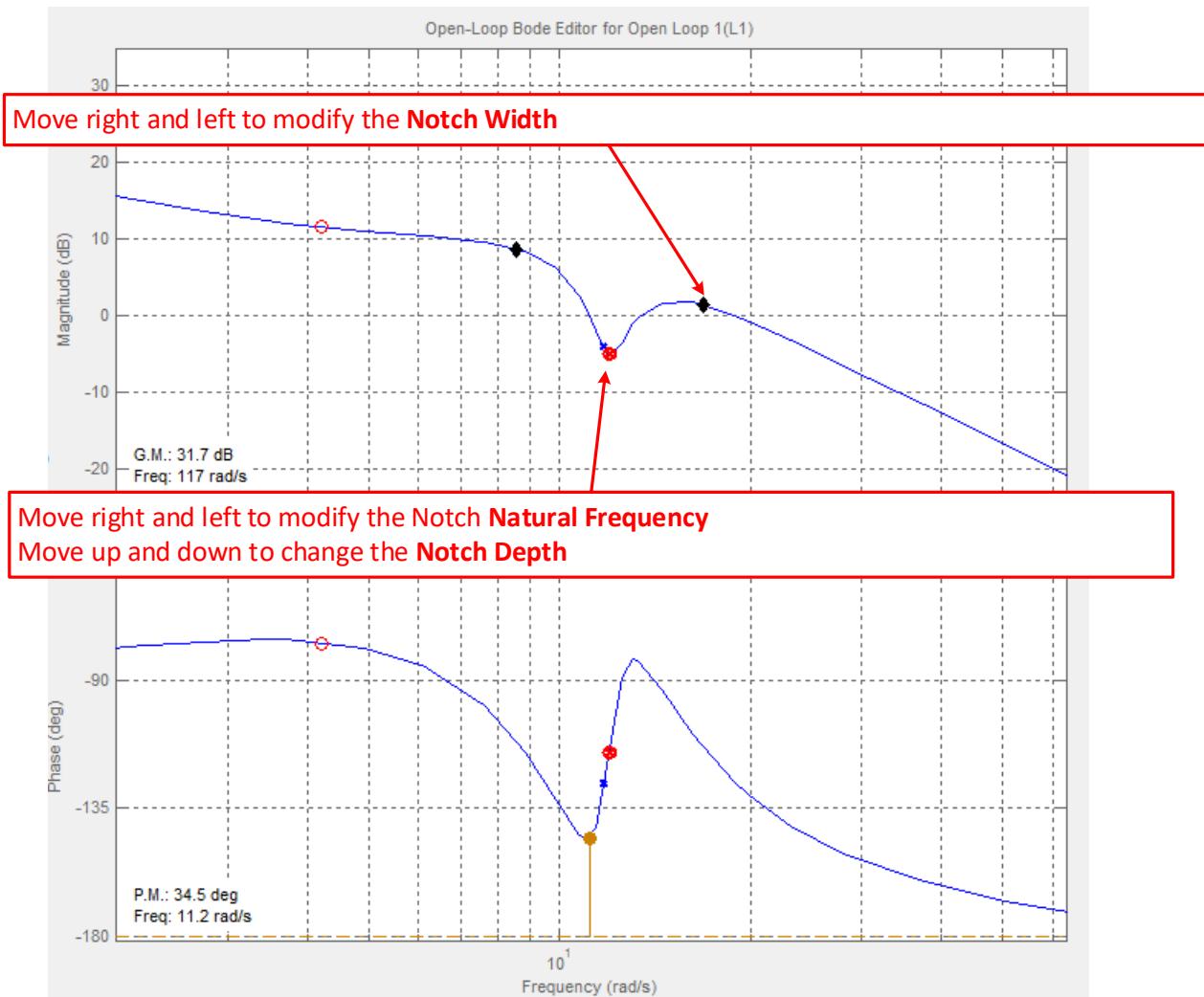


Figure 412: Adjusting the notch

While simultaneously viewing the Bode diagram of the closed-loop transfer function and the Bode diagram of the open-loop transfer function, adjust the three parameters so that the resonance on the Bode diagram of the closed-loop transfer function disappears.

Once again the settings affecting the performance of the closed-loop are substantially improved (Figure 413).

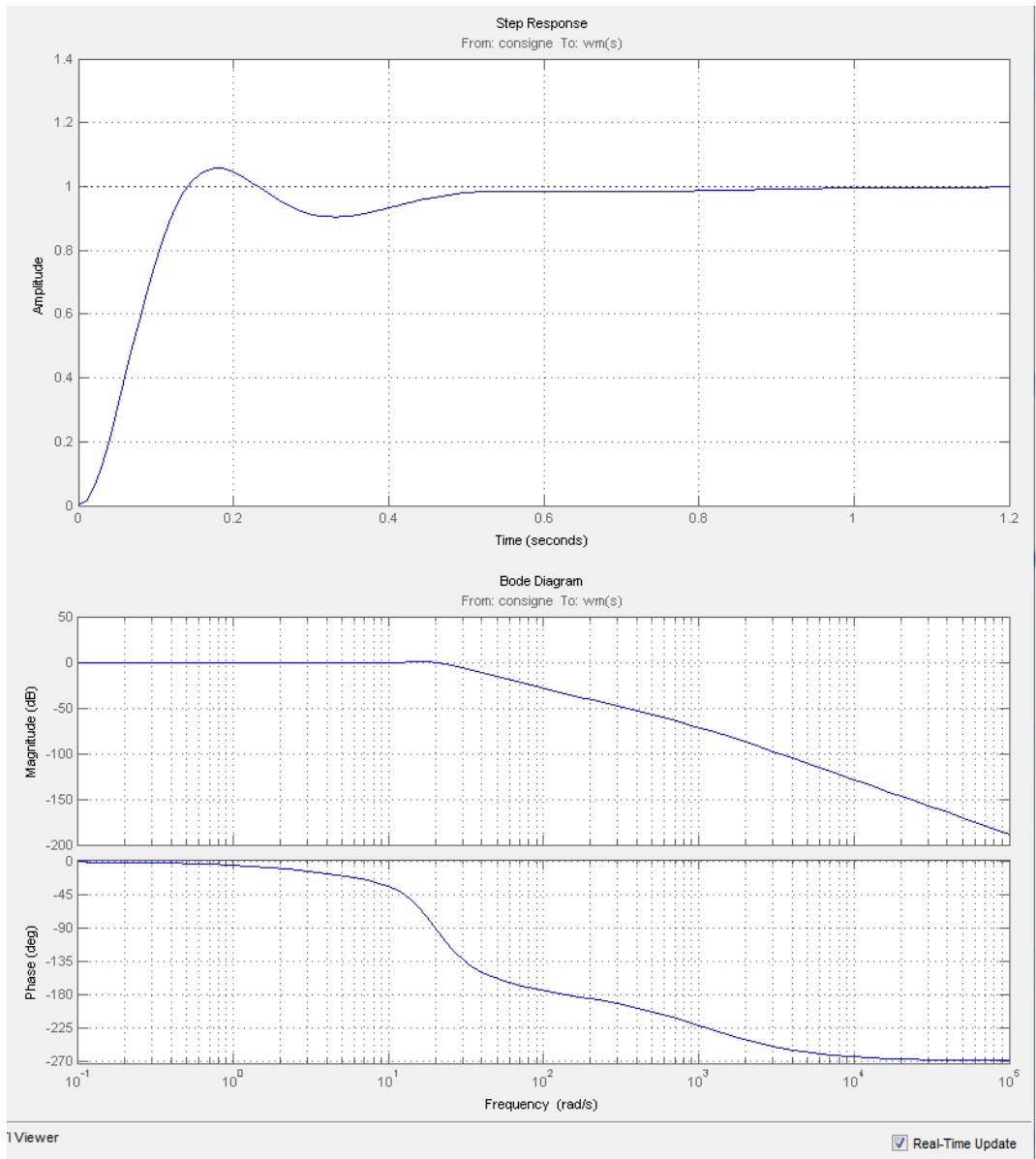


Figure 413: Performance of the closed-loop after adjusting the filter

The step response is accurate, fast and well damped. The compensator helped find a good compromise between all the required performances.

It is possible to view the compensator transfer function in the **Control and Estimation Tool Manager** window, **Compensator Design** tab and see the value of filter adjustment parameters that have led to improved performance.

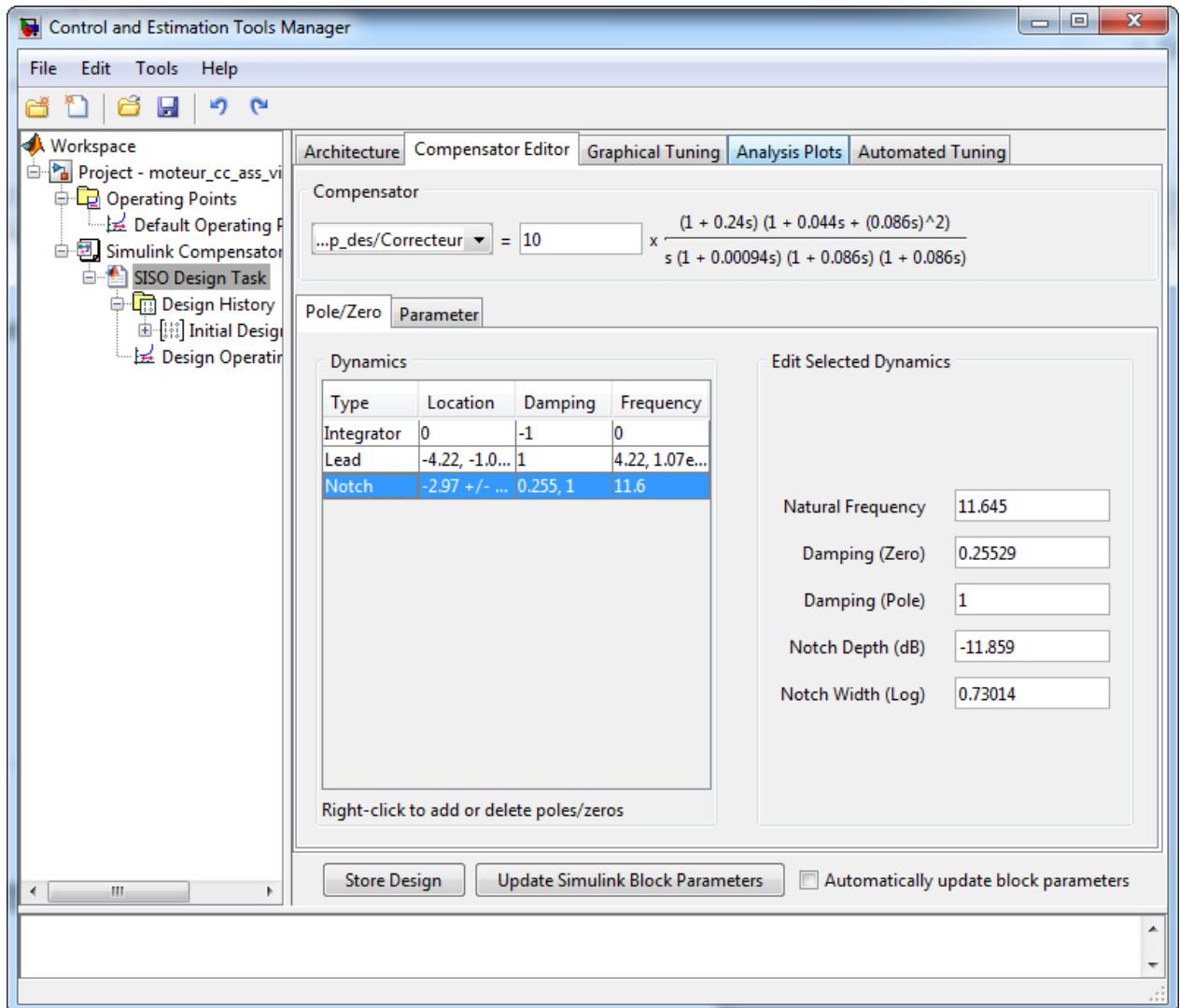


Figure 414: Viewing the compensator transfer function

It is also possible to see the influence of the rejecter filter on all of the curves representing the behavior of the open-loop.

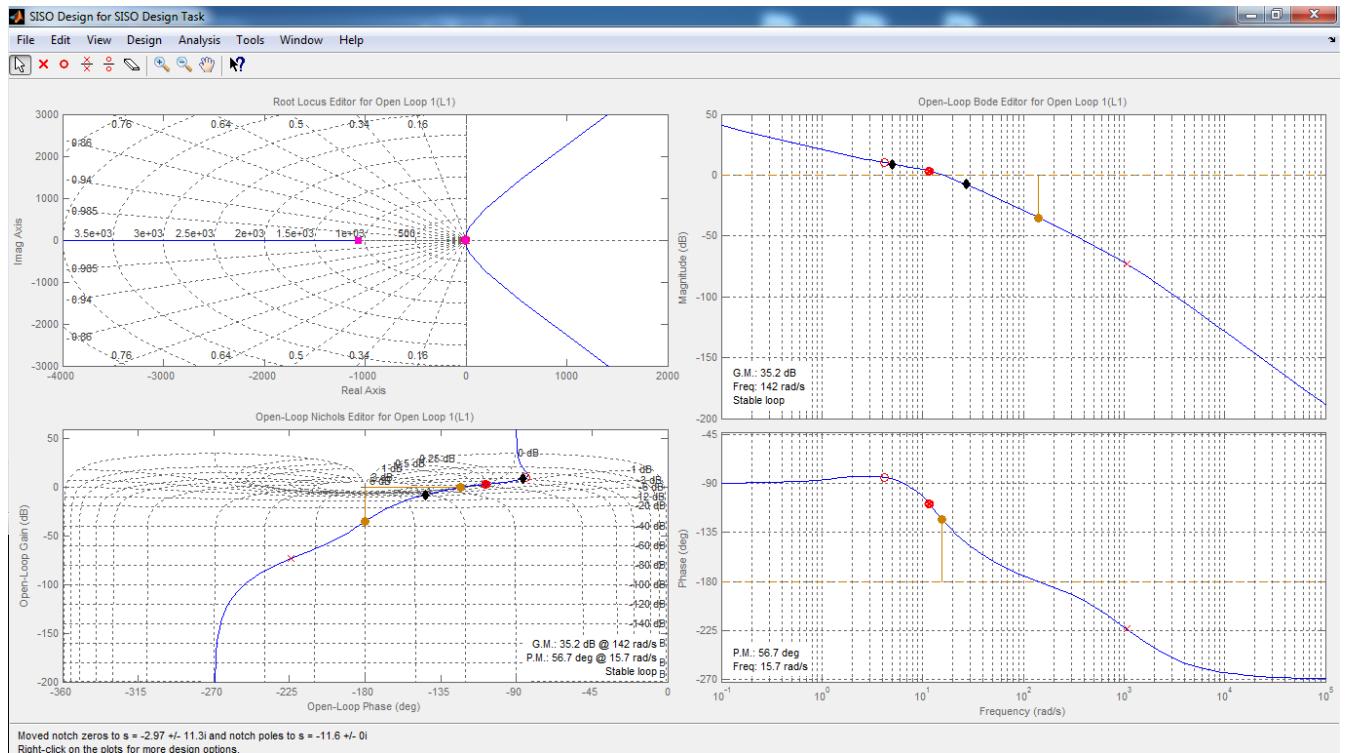


Figure 415: Influence of the filter on the behavior of the open-loop

10. Exporting the compensator transfer function to the Simulink model

Click on **Update Simulink Block Parameter** in the **Compensator Editor** window to export the compensator transfer function to the Simulink model.

Return to the Simulink model, launch the simulation and view the system response in the scope.

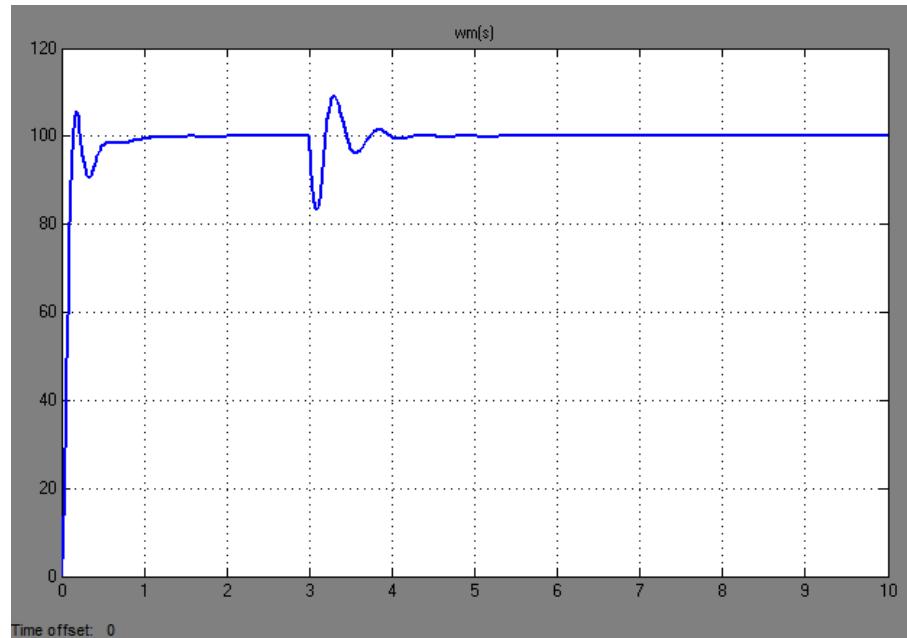


Figure 416: Corrected step response

The corrected step response (Figure 417) is accurate, the speed and depreciation are satisfactory and disturbance is rejected well.

The comparison of the corrected and non-corrected responses shows the contribution of the compensator.

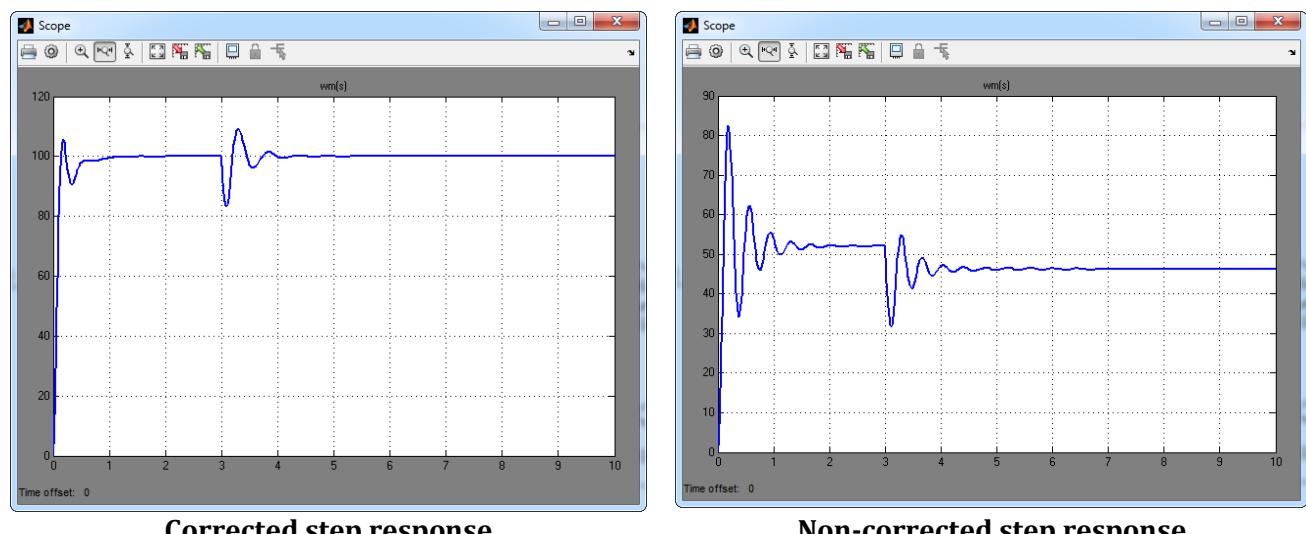


Figure 417: Comparison of corrected and non-corrected step responses

Appendix 1: Setting scopes

Open the file “**circuit_RL_scope_US.slx**” and **launch** the simulation.

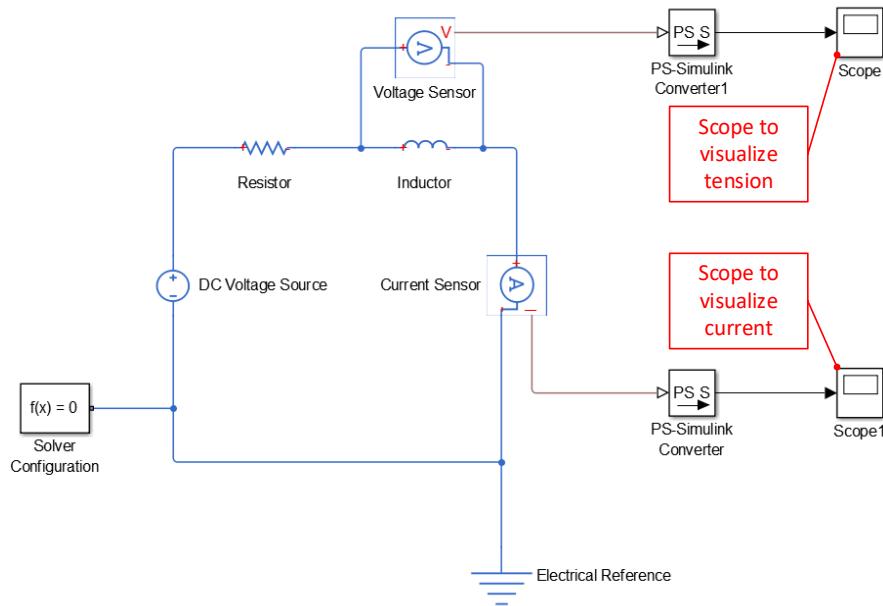


Figure 418: Scopes measuring the current in the circuit and the voltage across the coil terminals

Open the scope measuring the voltage across the coil terminals and view the appearance and formatting of the curve.

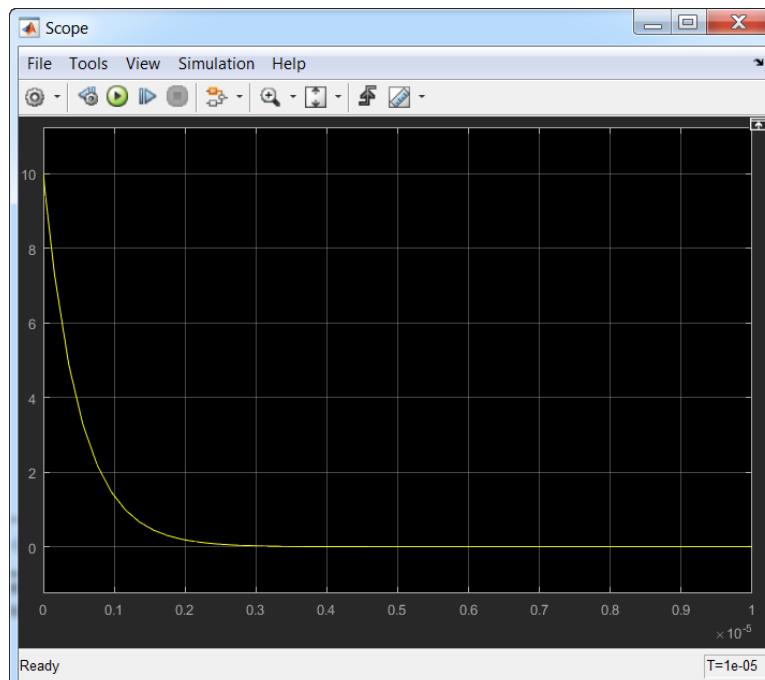


Figure 419: Measuring the voltage across the coil terminals

Upon opening the scope, the command bar is available in the upper part of the window (Figure 420).

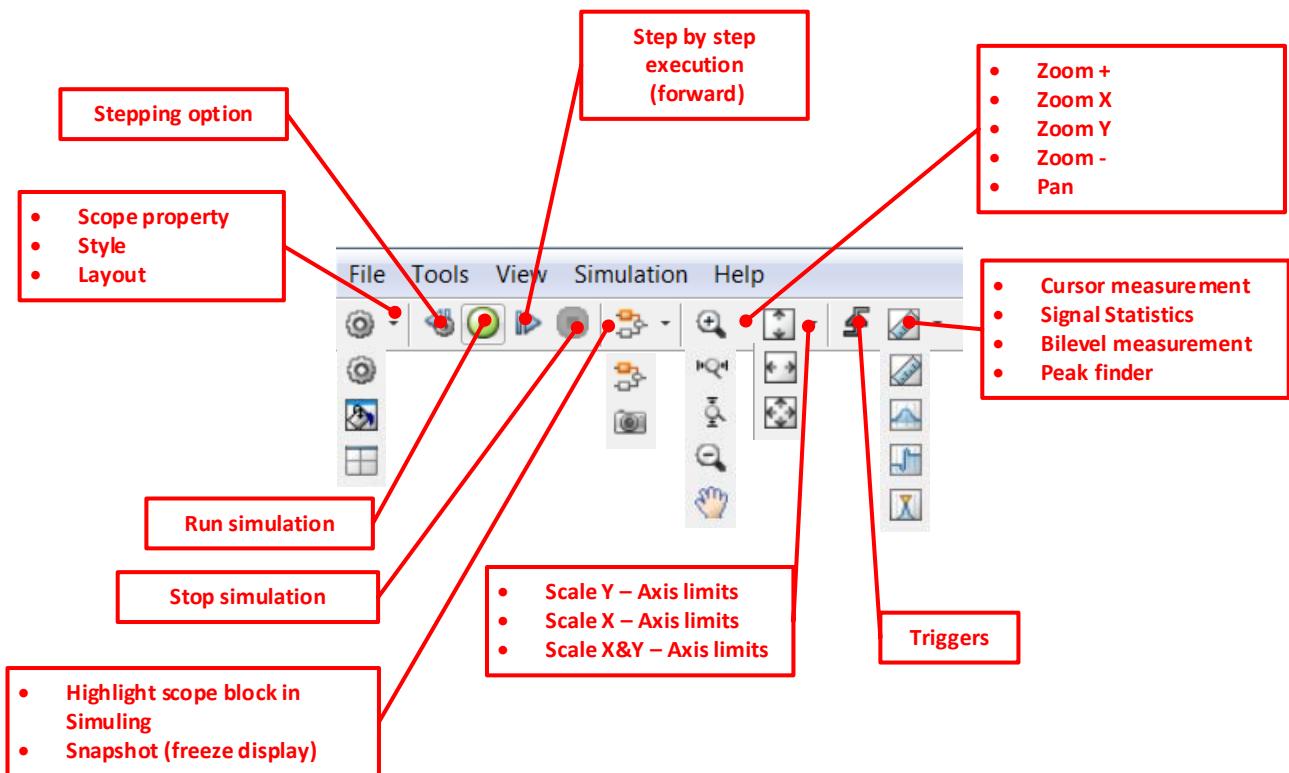


Figure 420: Scope command bar

The most useful command is the automatic scaling of axes . Think about using it at the end of a simulation when opening the scope to view the curve. It is possible that upon opening the scope the curve is not in the area displayed by the axes and nothing is visible on the scope.

You can change the formatting of the curves available by clicking on the icon from the **Configuration Properties** drop-down menu .

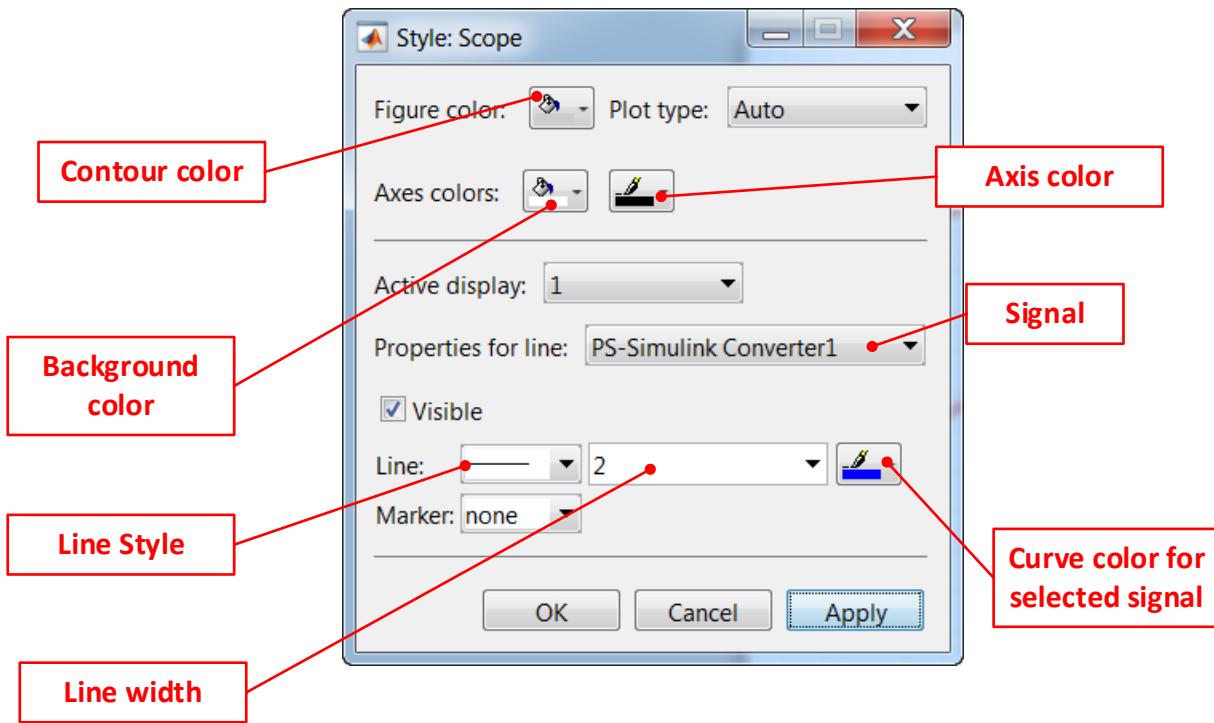


Figure 421: Formatting curves in the scope

Modify the formatting parameters as indicated in Figure 421 and see the results obtained (Figure 422).

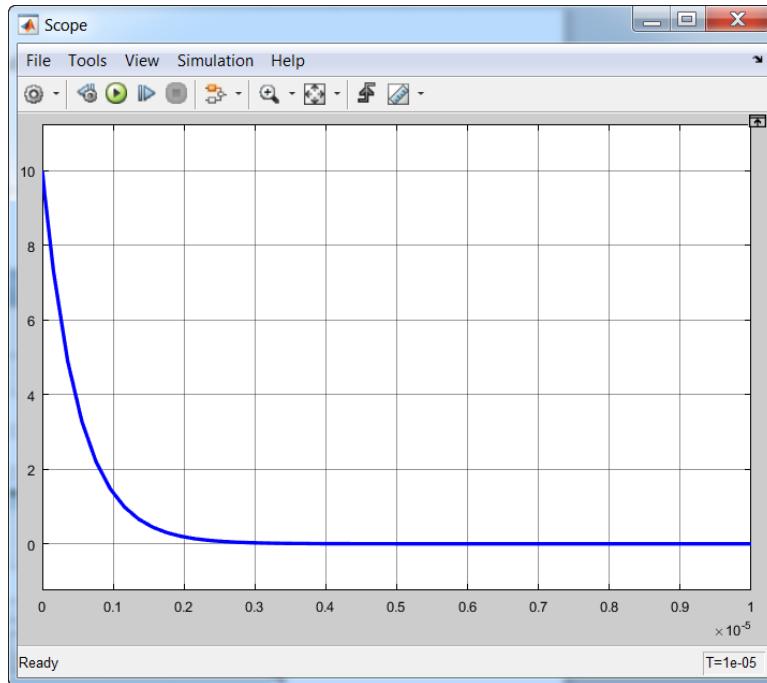


Figure 422: Changing the format of a curve in the scope

To display multiple curves in the same scope click on **Configuration Properties**  and change the **Number of input ports** field to 2 in order to define two inputs for the scope (Figure 423).

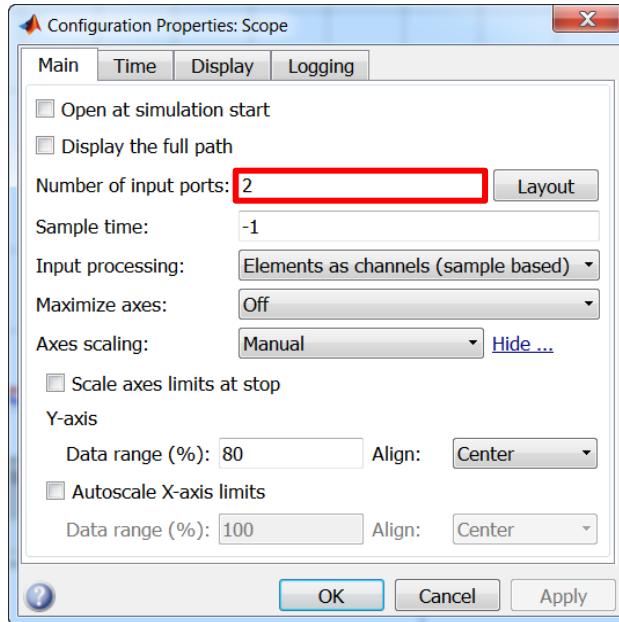


Figure 423: Changing the number of inputs of a scope

The scope now has 2 inputs. Connecting the second input to the signal measuring the current in the circuit as indicated in Figure 424.

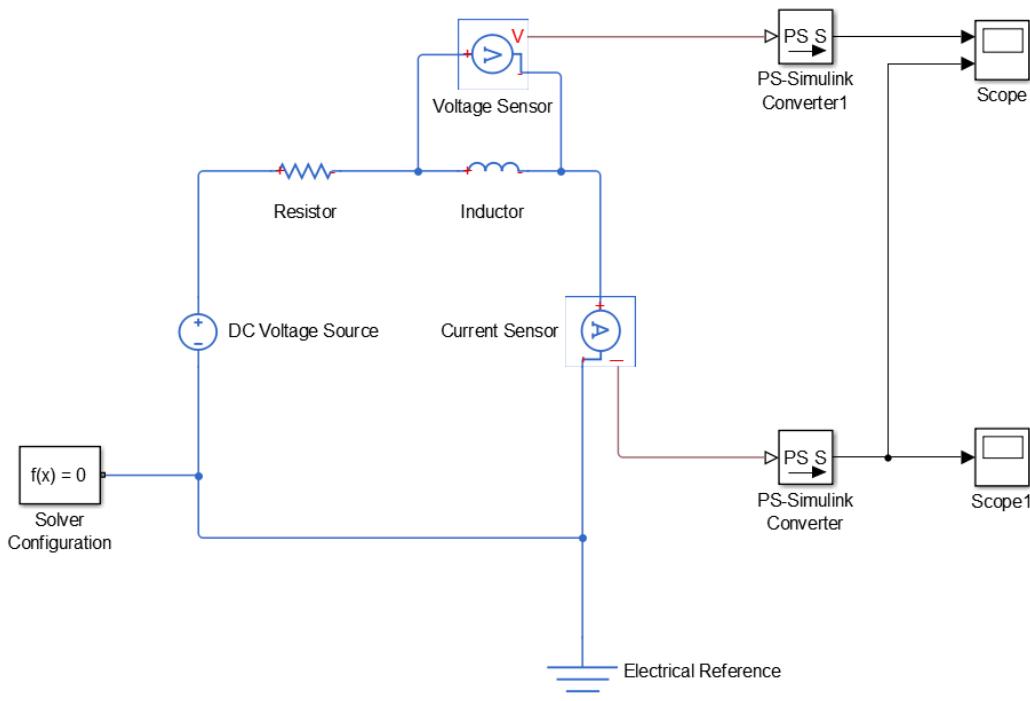


Figure 424: Connecting one scope to two inputs

Launch the simulation and open the scope to get the curve in Figure 425.

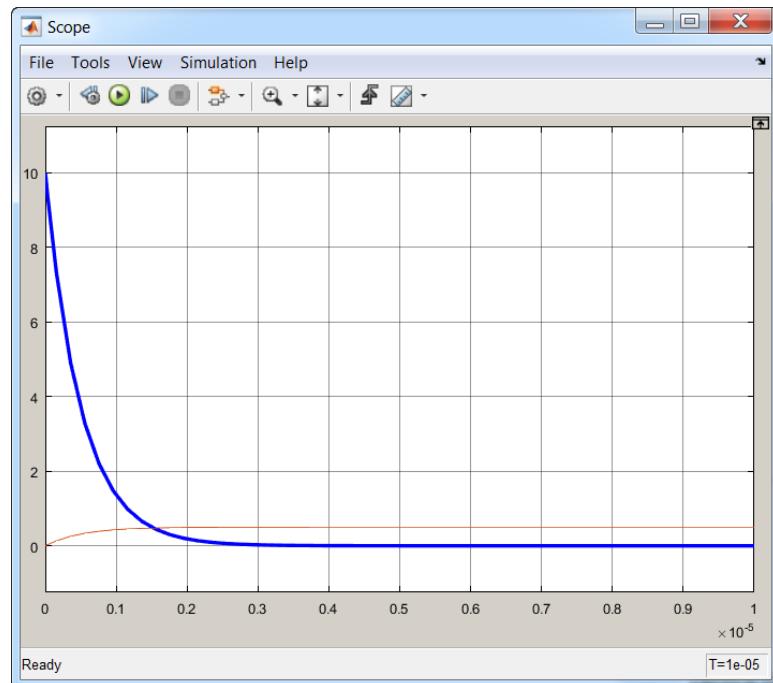


Figure 425: View of two curves in the same scope

Changing the characteristics of the second curve using the icon , as indicated in Figure 426.

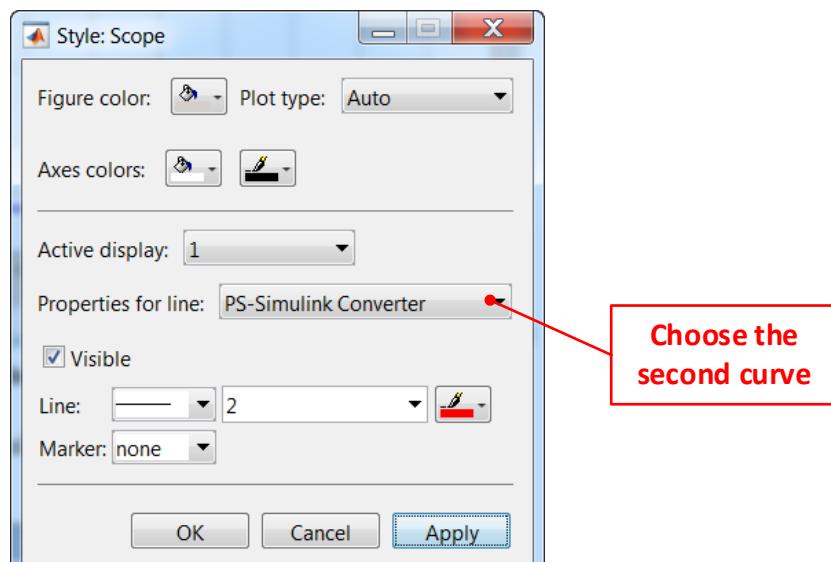


Figure 426: Changing the characteristics of the second curve

You should obtain the configuration of Figure 427.

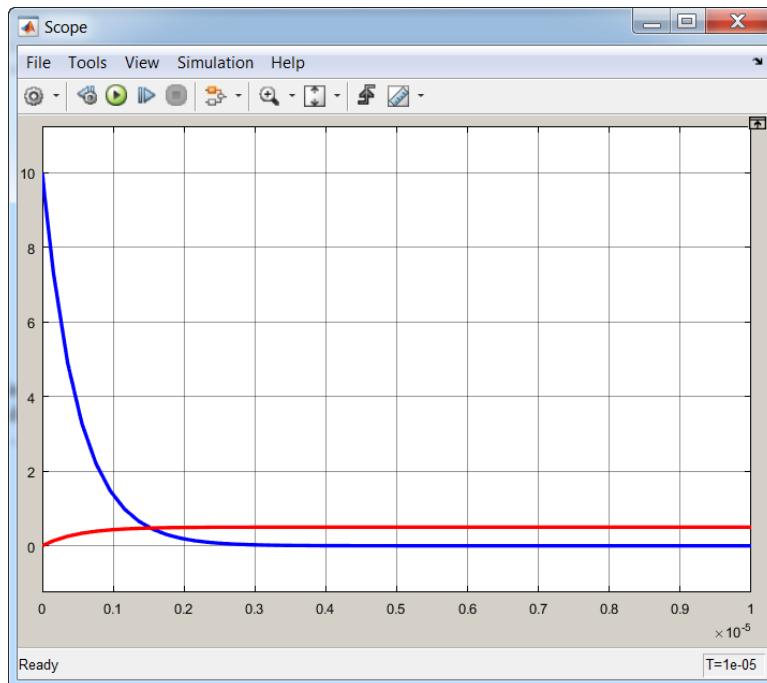


Figure 427: Changing the characteristics of the second curve

If you want to display the two curves on different graphs, click on the **layout** icon and choose the desired arrangement for the curves (Figure 428).

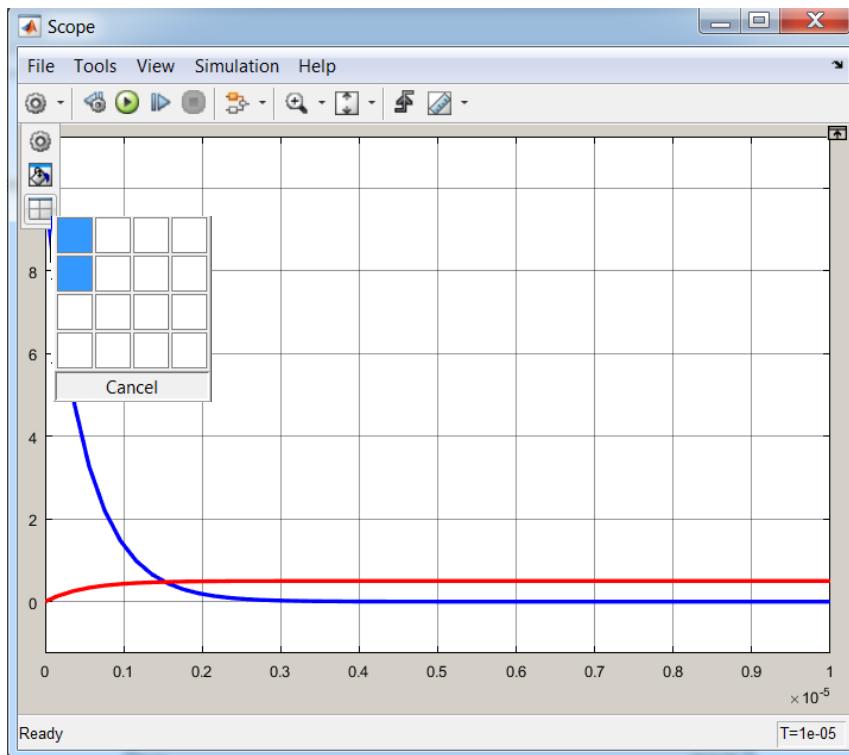


Figure 428: Changing the number of windows in the scope

After scaling the curves using the  command, we get the display shown in Figure 429.

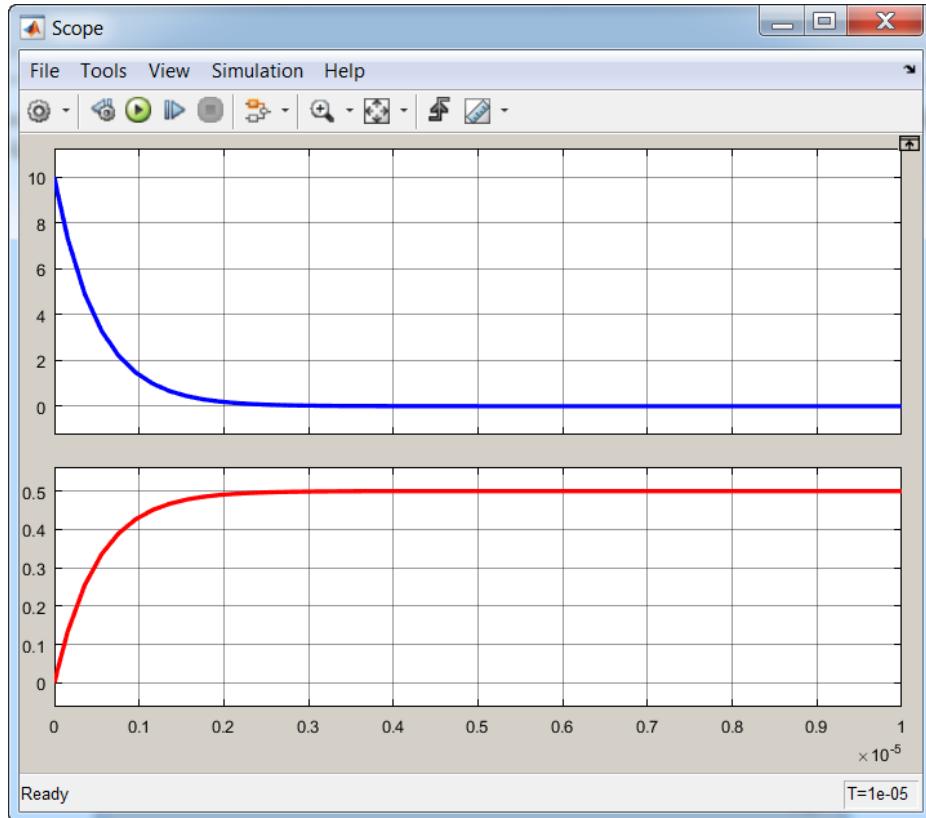


Figure 429: Displaying curves in a scope in multiple windows

A cursor tool is very convenient to use in the scope, to use it click on the **Cursor Measurement**  icon to make the cursors appear. It is then possible to have access to all relevant measurement parameters ΔT , ΔY , slope of the line connecting two cursors...(Figure 430).

For more information on the use of scopes you can use the MATLAB help.

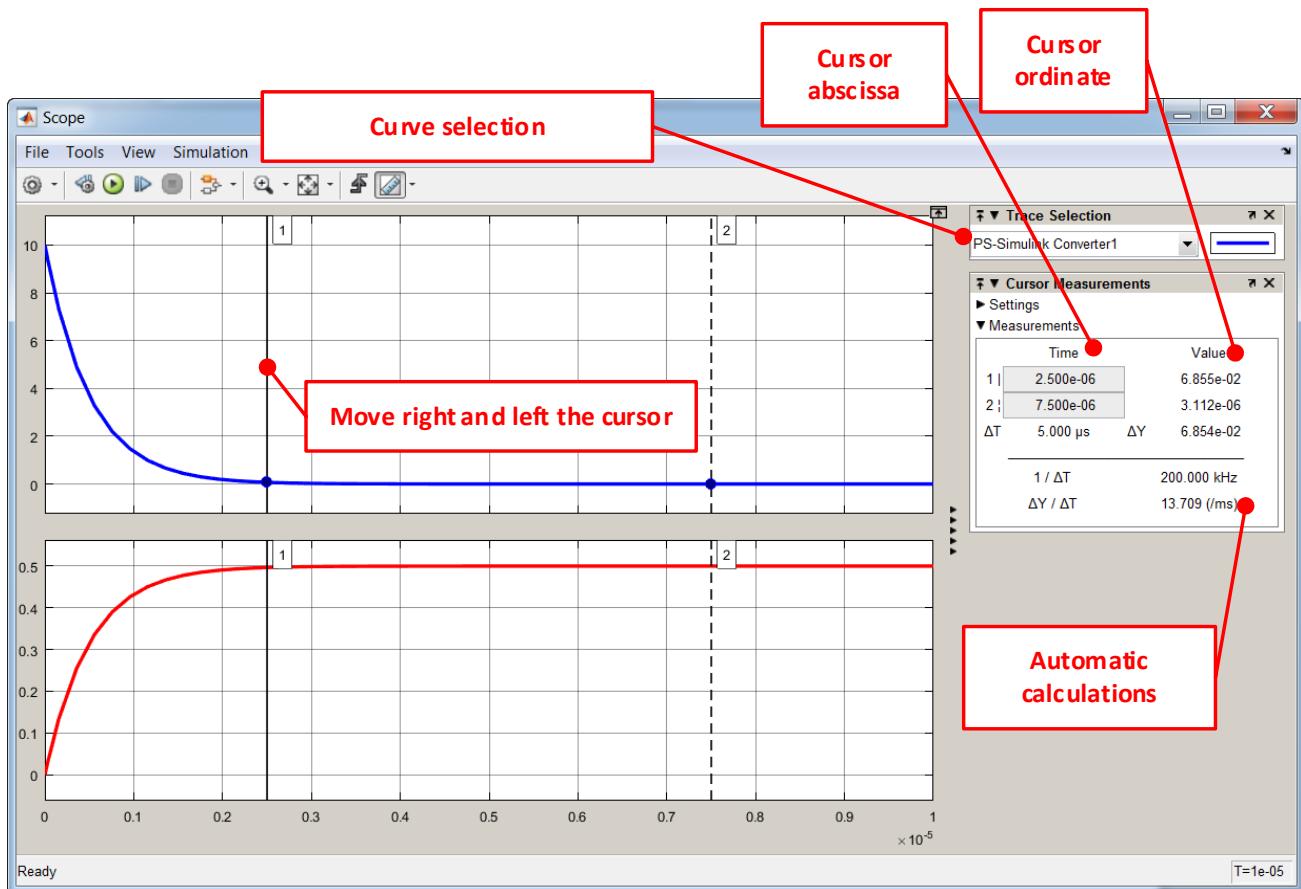
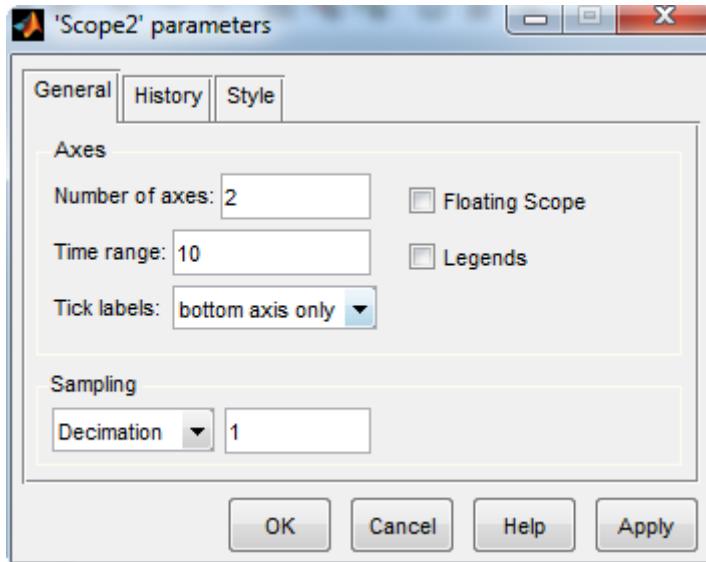


Figure 430: Using cursors in a scope

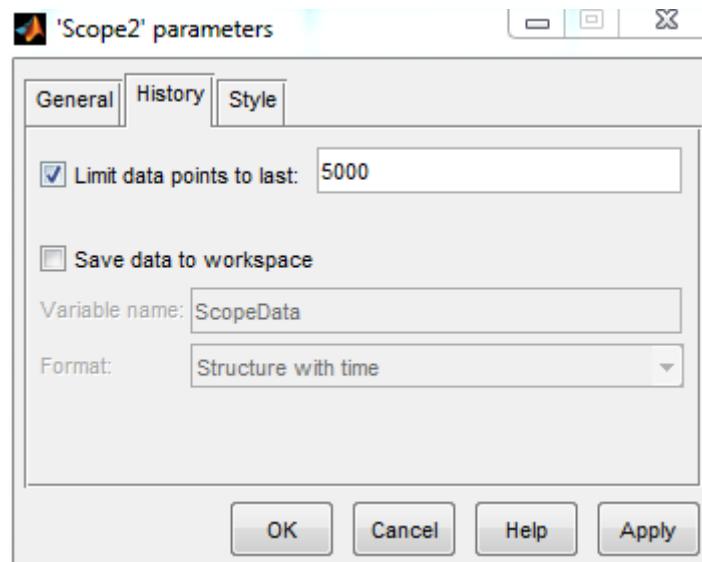
To set a scope with multiple inputs, double-click on the scope and click on **Parameters** .



In the **General** tab, choose the number of inputs of the Scope in the **Number of axes** field.

It is also possible to request the display of a legend by checking the **Legends** box.

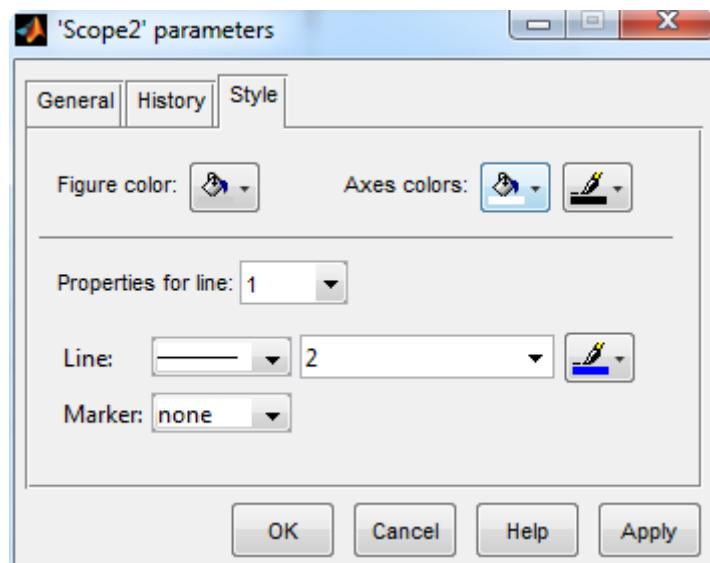
In the **History** tab, it is possible to specify the maximum number of points that the scope will record. It is often necessary to uncheck this box in order to be able to see all of the results from a simulation.



From this tab you can also save the results of the simulation in the **workspace** in the form of a matrix that can be used with **MATLAB**.

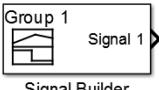
To do this, check the **Save data to workspace** box and indicate the variable name and its format (by default **Structure with time** saves both the values of all points of the signals, but also information such as the name of the signal, the size of the matrices...)

In the **Style** tab, you can choose the display and background colors of the curves, add markers, change the line thickness...



Appendix 2: Using Signal Builder

“Signal Builder” is a signal generator that can be set manually. This block is very useful for imposing command laws.

Component function	View	Library
Configurable signal source		Simulink/Sources

Drag and drop a “Signal Builder” block from the **Simulink** library and **open** this block.

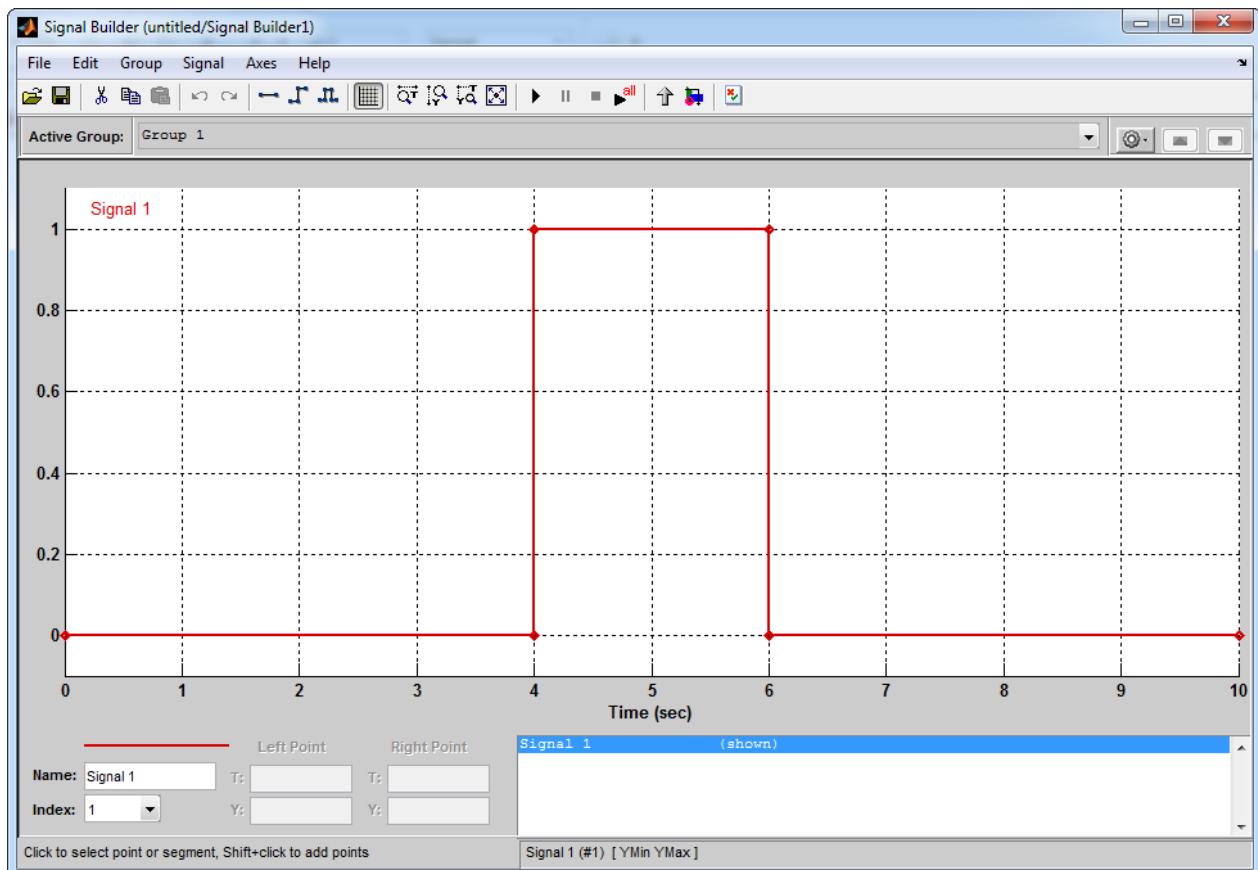


Figure 431: Signal Builder settings window

The signal builder command window gives access to numerous functions.

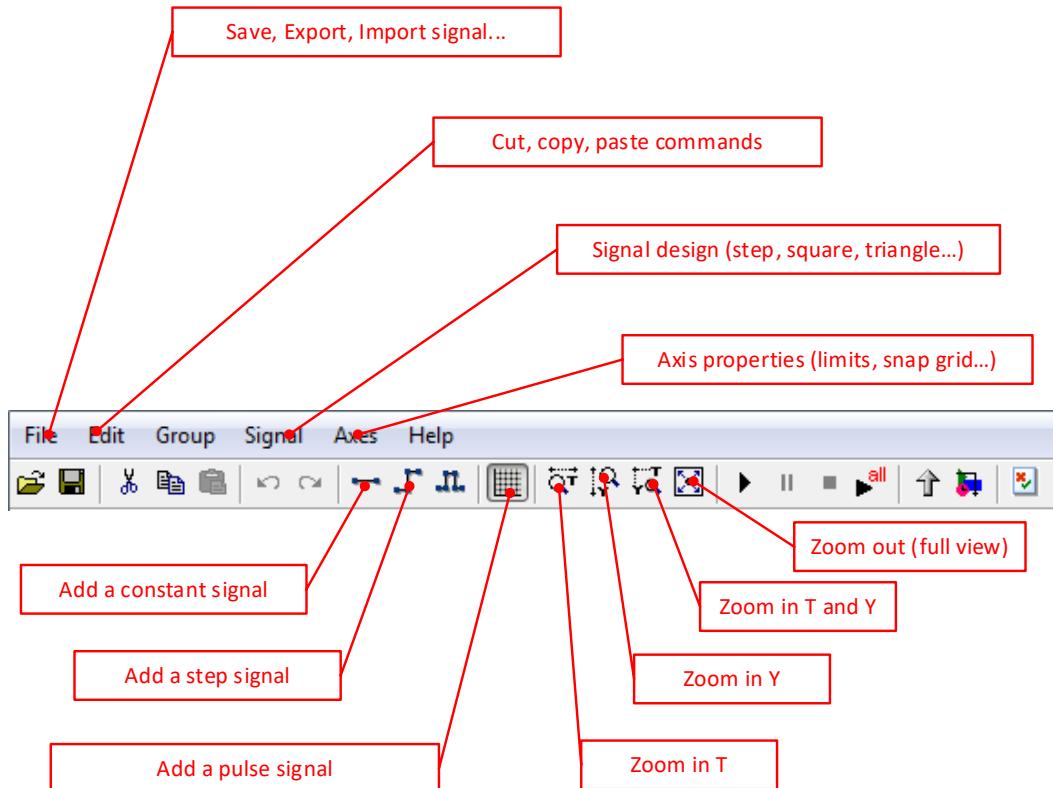


Figure 432: Signal Builder command bar

First you must specify the duration of the signal. For this select **Signal/Change Time Range**, accessible from the command bar.

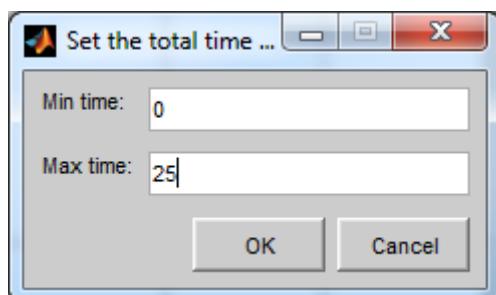


Figure 433: Choosing the duration of the signal in "Signal Builder"

Choose a signal duration of 25s.

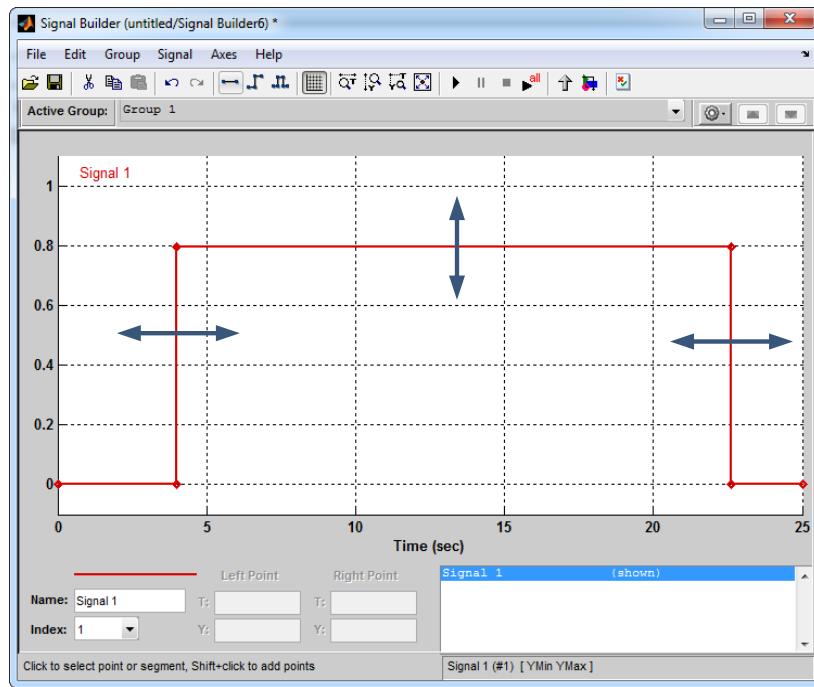


Figure 434: Manually moving segments in the “Signal Builder”

You can manually move the segments that compose the signal.

With the mouse cursor and using **drag and drop**, horizontally move the vertical segments and vertically move the horizontal segments.

While moving, “left point” and “right point” can be displayed in the boxes located under the signal segment position.

Simply left-click on a segment to select it, and you obtain the exact positions in the “left point” and “right point” boxes.

It is possible to move only one point to obtain ramp type signals.

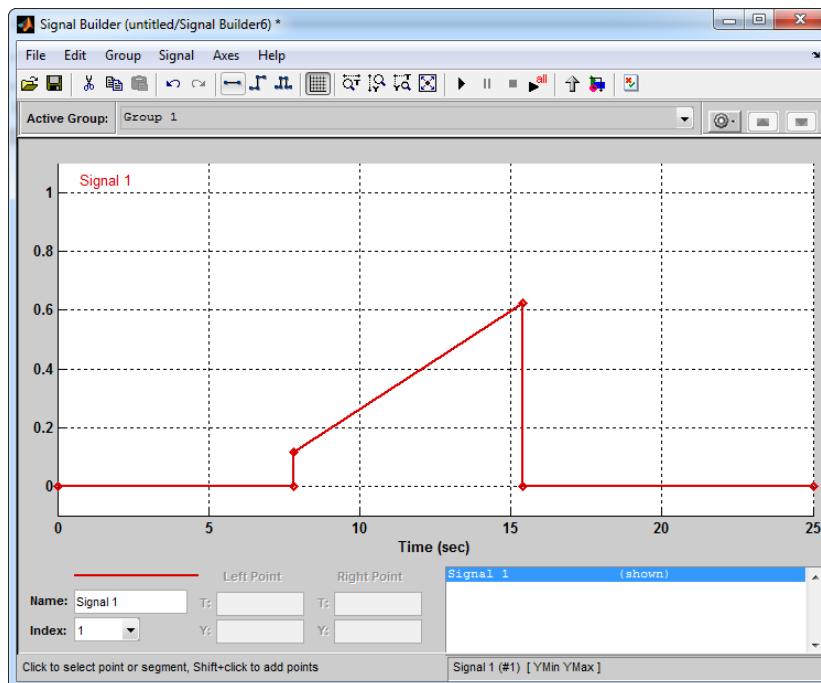


Figure 435: Moving one point in the “Signal Builder”

The principle of constructing any signal is to create points that will be connected by segments that can then be moved freely.

To create new points, use **shift+left-click** and place the new points directly on the signal.

Approximately **place** new points in accordance with Figure 436.

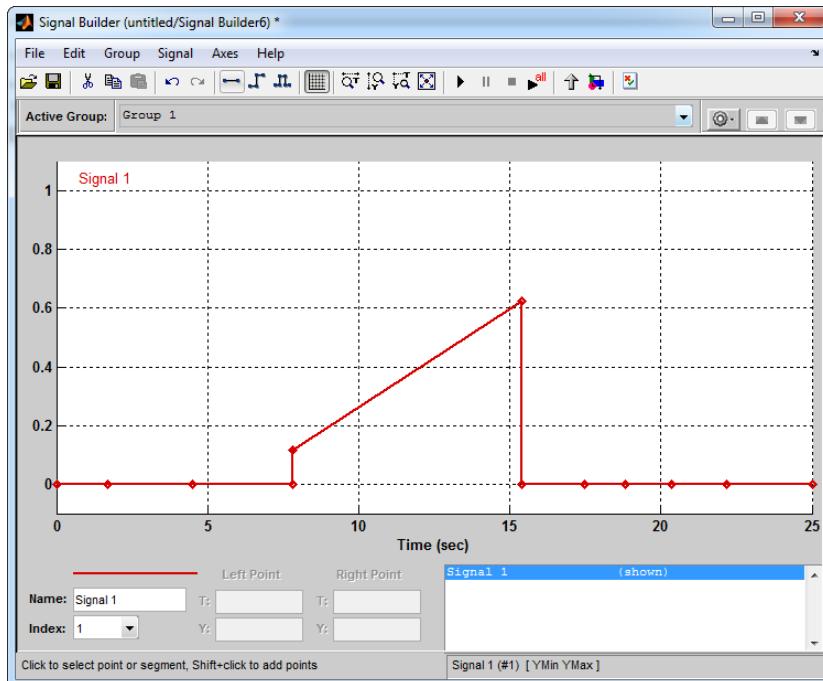


Figure 436: Placing new points on the signal

Moving the points and segments to approximately obtain the signal of Figure 437.

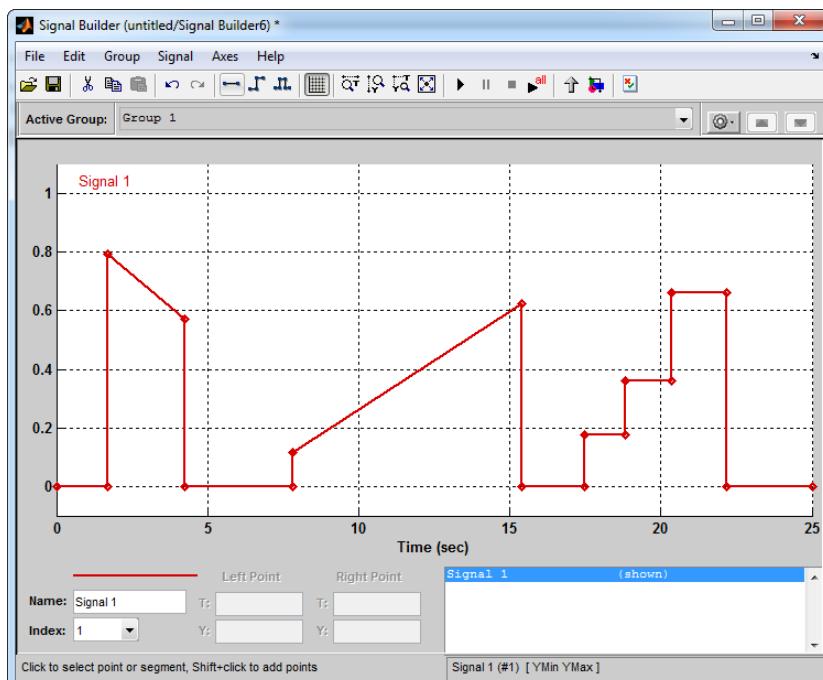
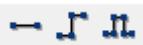


Figure 437: Formatting the signal by moving segments and points

It is also possible to create a block to generate more signals.

Click on one of the three available signals  from the command bar and choose to introduce a new signal.

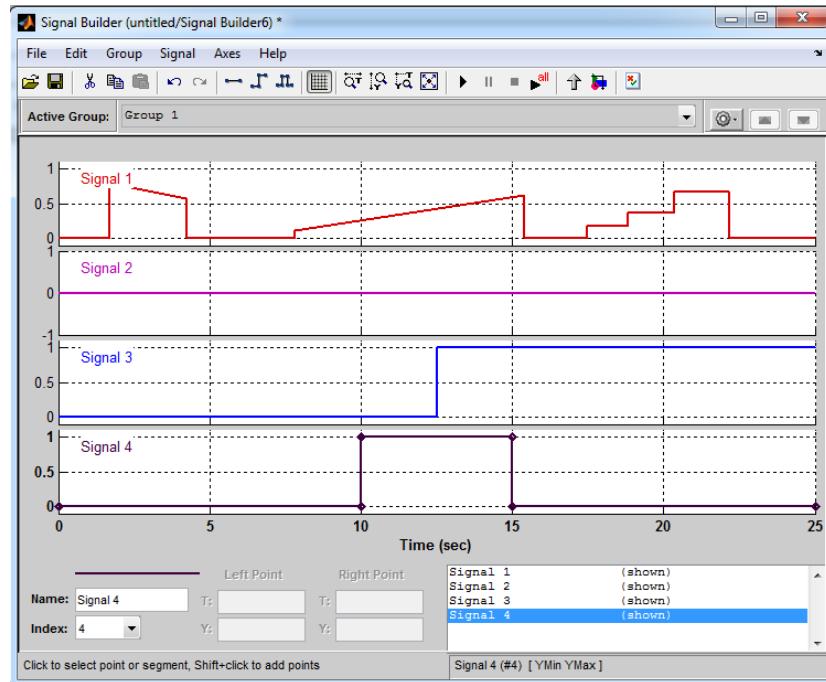


Figure 438: Generating many signals

The block now has 4 outputs.

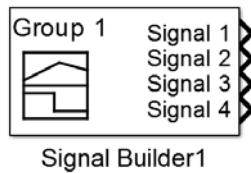


Figure 439: View of multiple outputs in the "Signal Builder"

Table of Figures

Figure 1: Simplified V-Model.....	11
Figure 2: Competences of the engineer.....	12
Figure 3: Specifications/real system/ model triptyc.....	13
Figure 4: material architecture required to implement the procedure	14
Figure 5: Analysis and Requirements phase	15
Figure 6: position servo controller requirements diagram.....	15
Figure 7: Design, Modeling, Simulation Phase.....	16
Figure 8: input/output relationship for a DC motor.....	17
Figure 9: equations for the behavior of DC motors.....	17
Figure 10: modeling the DC motor in the form of a block diagram using Simulink.....	18
Figure 11 : "White box" modeling of the DC motor.....	18
Figure 12: model created through assembling the components of the DC motor	19
Figure 13: acausal multi-physics modeling of the DC motor	19
Figure 14: resolution and visualization of results	20
Figure 15: measurement of the evolution of the rotation speed of the real motor as a function of time	20
Figure 16: evaluation of differences between simulated and measured performances.....	21
Figure 17: trial conducted to estimate unknown parameters	22
Figure 18: illustration of the principal of parameter estimation.....	22
Figure 19: response of the model before and after estimation of unknown parameters.....	23
Figure 20: modeling the servo controller for the motor position	23
Figure 21: DC motor position response without correction	24
Figure 22: illustration of the use of a command control tool for adjusting a compensator	25
Figure 23: DC motor position response with PID correction	25
Figure 24: Model-in-the-loop (MIL).....	25
Figure 25: Coding Implementation Phase.....	26
Figure 26: structure of the model at the end of the Design Modeling Simulation phase	26
Figure 27: Software-in-the-loop (SIL)	27
Figure 28: Processor-in-the-loop	28
Figure 29: Integration and Testing Phase.....	28
Figure 30: materials architecture for hardware-in-the-loop in external mode	29
Figure 31: hardware-in-the-loop in external mode: integration of materials into the loop	29
Figure 32: materials architecture for Hardware-In-the-Loop in embedded mode	30
Figure 33: Hardware-In-the-Loop embedded mode: autonomous functioning of materials	30
Figure 34: View of the speed of the motor output at the end of the Hardware-in-the-loop phase	31
Figure 35: Validation Receipt Phase.....	31
Figure 36 : example of script written in MATLAB language	32
Figure 37: the result obtained after execution of the script	33
Figure 38: example of Simulink modeling	33
Figure 39: obtained results visualized in a scope	34
Figure 40: example of Simscape modeling	34
Figure 41: obtained results visualized in a scope	35
Figure 42: Simscape and its libraries	36
Figure 43: modeling of the behavior of an automatic transmission with Stateflow	37
Figure 44: MATLAB for multi-physics modeling	38
Figure 45: MATLAB environment window	39
Figure 46: MATLAB environment command bar	40
Figure 47: Simulink library	40
Figure 48: Simulink environment window	41
Figure 49: the MATLAB path.....	42
Figure 50: adding a file to the "Path" for the current session	43
Figure 51: Energy Chain/Information chain diagram	44
Figure 52: correspondence between the energy chain / information chain diagram and MATLAB tools.....	45
Figure 53: photo of the automatic pilot	46
Figure 54: relationship between the energy chain/information chain diagram and the MATLAB tools	47
Figure 55: copy of the multi-physics modeling screen of the boat automatic pilot executed with MATLAB	48
Figure 56: View of the course instruction.....	49
Figure 57: SimMechanics visualization window.....	50
Figure 58: SimMechanics visualization bar	50
Figure 59: SimMechanics visualization window	51
Figure 60: SimMechanics time bar	51

Figure 61: View of the course instruction and of the effective course followed by the boat.....	52
Figure 62: View of the relative dimensions of the motor.....	52
Figure 63: View of the rotation angle of the helm	53
Figure 64: View of pressures in the chambers in front and behind the cylinder	53
Figure 65: View of the flow rates in the chambers in front and behind the cylinder.....	54
Figure 66: MATLAB – Simulink model of the autopilot information chain	55
Figure 67: modeling the course control with state diagrams	55
Figure 68: Simscape model of the electrical portion of the autopilot energy chain.....	56
Figure 69: SimHydraulics model of the hydraulic part of the autopilot energy chain.....	57
Figure 70: SimMechanics model of the autopilot structure.....	58
Figure 71: Simulink modeling of dynamic behavior of the boat.....	59
Figure 72: model of the hydraulic pilot with interactive piloting	60
Figure 73: using interactive piloting on a model.....	61
Figure 74: presentation of the Maxpid robot	62
Figure 75: multi-physics model of the MAXPID robot	64
Figure 76: position of the MAXPID robot arm	65
Figure 77: Visualization of the control voltage for the motor.....	65
Figure 78: Visualization of the current in the motor armature.....	66
Figure 79: Control'X linear axis.....	67
Figure 80: multi-physical model of the Control'X system.....	68
Figure 81: Visualization of the speed and the position of the linear axis.....	69
Figure 82: View of the command voltage of the motor and the current in the armature	69
Figure 83: evaluation of the difference in performance between the model performance and those of the actual axis.....	70
Figure 84: Evaluation of the differences between model/reality for speed and axis position.....	71
Figure 85: Evaluation of the differences between model/reality for armature current and command pressure	71
Figure 86: R-L circuit	73
Figure 87: components required for modeling the R-L circuit with Simscape.....	74
Figure 88: useful commands	75
Figure 89: Simscape Model of R-L circuit	76
Figure 90: useful commands	76
Figure 91: different port types in Simscape.....	76
Figure 92: identification of the connections in Simscape model	77
Figure 93: evolution of the intensity of current in an RL circuit	81
Figure 94: evolution of the voltage at the coil terminals in an RL circuit.....	82
Figure 95: the components required for modeling the R-L circuit with Simscape.....	83
Figure 96: R-L circuit model created with Simulink.....	84
Figure 97: Useful commands	84
Figure 98: evolution of the intensity of current in an RL circuit	86
Figure 99: advantages and disadvantages of the causal and acausal approaches	87
Figure 100: PCP ports in Simscape components	88
Figure 101: References for Simscape physical fields	89
Figure 102: the Across and Through variables in Simscape	90
Figure 103: source of force oriented positively	92
Figure 104: source of force oriented negatively	92
Figure 105: orientation of a source of force	93
Figure 106: View of the influence of the orientation of the source of force	93
Figure 107: example of measuring electrical parameters	94
Figure 108: placing of sensors	95
Figure 109: View of the voltage across the coil for the two sensor orientations	96
Figure 110: View of the intensity of current in the circuit according to the orientation of the sensors	96
Figure 111: solver settings window	97
Figure 112: photo of the linear axis	99
Figure 113: the components required for modeling the linear axis in Simscape	100
Figure 114: view of the physical fields involved in modeling the linear axis with Simscape	101
Figure 115: Simscape model of the linear axis	102
Figure 116: useful commands	102
Figure 117: specification of the Simscape variables to be taken into account in the logger	113
Figure 118: Simscape logger window	114
Figure 119: creation of a sub-system	116
Figure 120: useful commands	116
Figure 121: creation of a sub-system	117
Figure 122: View of the contents of the sub-system	117
Figure 123: renaming the ports in a sub-system.....	118
Figure 124: navigate in the sub-systems of a model	118
Figure 125: Simscape model of the linear axis with "motor" sub-system.....	119
Figure 126: Simscape model of the linear axis without speed sensor	119

Figure 127: Simscape model of the linear axis with name of signals	120
Figure 128: Simscape model of the linear axis, creation of the “Axis” sub-system.....	120
Figure 129: Simscape model of the linear axis with “motor” and “Axe” sub-system.....	120
Figure 130: renaming the ports in a sub-system.....	121
Figure 131: finished Simscape model of the linear axis.....	121
Figure 132: configuration of sub-system mask window	122
Figure 133: displaying an image on a sub-system.....	122
Figure 134: response position of free running linear axis.....	123
Figure 135: model of the servo controller in linear position on the axis	123
Figure 136: components to add to a model.....	124
Figure 137: configuration of the adding block	125
Figure 138: response of the dedicated axis in position.....	126
Figure 139: template for loop in linear axis position	126
Figure 140: configuration of the PID block	127
Figure 141: corrected response of the loop in linear axis position	128
Figure 142: the components required for modeling the command from a hydraulic cylinder.....	129
Figure 143: view of the physical fields involved in modeling the hydraulic cylinder control	130
Figure 144: Simscape model of the hydraulic cylinder control.....	131
Figure 145: evolution of the speed and the position of the cylinder shaft	137
Figure 146: Simscape model of the hydraulic cylinder control with signals routing.....	138
Figure 147: Simscape model of a hydraulic cylinder control with a flow source.....	141
Figure 148: configuration of a pressure limiter block	142
Figure 149: variation in the opening of the valve depending on the pressure	142
Figure 150: evolution of the speed and position of the cylinder shaft	143
Figure 151: evolution of the speed and position of the cylinder shaft	143
Figure 152: PWM control principle.....	144
Figure 153: Illustration of how the “Controlled PWM Voltage” block functions	145
Figure 154: Controlled PWM Voltage block control voltage.....	145
Figure 155: PWM signal based on the control voltage	146
Figure 156: configuration of the Controlled PWM Voltage Source block.....	147
Figure 157: PWM control for a DC motor	148
Figure 158: rotation speed of the motor	148
Figure 159: rotation speed of the motor after changing the PWM frequency	149
Figure 160: PWM control of a DC motor with an H-Bridge	150
Figure 161: rotation speed of the motor controlled by the H-Bridge	150
Figure 162: average voltage of the motor power supply	151
Figure 163: configuration of the Controlled PWM Voltage block	152
Figure 164: configuration of the H-Bridge block.....	154
Figure 165: components of the Simulink “Dashboard” library	156
Figure 166: the Simulink Real-Time Pacer library	156
Figure 167: refreshing the Simulink library	157
Figure 168: interactive model navigation.....	158
Figure 169: view of the linear axis position in real time	158
Figure 170: “controlled_linear_axis_dashboard_start_US.slx” model.....	159
Figure 171: insertion of Dashboardlibrary blocks	160
Figure 172: configuration window of the knob block.....	161
Figure 173: appearance of the knob button after configuration and connection with the model	161
Figure 174: configuration window of the Dashboard Scope block.....	162
Figure 175: simulation of an interactive model with the components of the Dashboard library	163
Figure 176: introduction to the buck converter	164
Figure 177: operating principle of the buck converter	165
Figure 178: buck converter diagram.....	165
Figure 179: diagram of a current motor controlled by a buck converter.....	166
Figure 180: circulation of the current in the circuit in active phase and in freewheel phase	166
Figure 181: View of the effect of the duty cycle on the current ripple.....	167
Figure 182: View of the effect of the chopping frequency on the current ripple.....	167
Figure 183: View of the effect of the inductance value on the current ripple	168
Figure 184: analogy between the Simscape model form and the electrical diagram.....	169
Figure 185: Simscape buck converter model	170
Figure 186: components required for modeling the buck converter	170
Figure 187: configuring the Diode block	172
Figure 188: configuring the MOSFET block	173
Figure 189: configuring the Pulse Generator block	174
Figure 190: evolution of the rotation speed of the motor controlled by a buck converter	175
Figure 191: creating a sub-system to facilitate the model as a learning tool	176
Figure 192: installing current sensors onto the circuit.....	176

Figure 193: adding a sensor to the model	177
Figure 194: an instrumented model facilitated for teaching.....	178
Figure 195: supply_current sub-system.....	178
Figure 196: change in the current in the three branches of the circuit.....	179
Figure 197: Model not adapted for teaching	181
Figure 198: The teaching model.....	181
Figure 199: improving the teaching model.....	182
Figure 200: the optimized teaching model.....	183
Figure 201: adjusting the commutation period of the transistor.....	184
Figure 202: View of the average and instantaneous value of the motor current.....	185
Figure 203: addition of a smoothing inductance in series with the motor.....	186
Figure 204: use and view of circulation of current in the buck converter.....	187
Figure 205: adjustments to the simulation parameters	188
Figure 206: use and view of the effect of the duty cycle on the motor current.....	189
Figure 207: adjustments to the simulation parameters	190
Figure 208: use and visualization of the effect of the chopping frequency on the motor current.....	191
Figure 209: adjustments to the simulation parameters	192
Figure 210: View and use of the effect of load inductance on the motor current.....	193
Figure 211: table of data.....	196
Figure 212: plot of two vectors.....	197
Figure 213: plot of two curves in the same graphics window	198
Figure 214: displaying the grid on a graphic.....	198
Figure 215: opening a new graphics window	199
Figure 216: codes for shaping curves	199
Figure 217: shaping a curve: modifying line color and style.....	200
Figure 218: shaping a curve: adding markers	200
Figure 219: shaping a curve: choosing line thickness.....	201
Figure 220: shaping a curve: legend and axis annotation.....	202
Figure 221: shaping a curve: modifying axis scales.....	202
Figure 222: Editor window for editing scripts	203
Figure 223: first script with MATLAB.....	203
Figure 224: window showing automatic addition of the folder to the MATLAB “path”	204
Figure 225: Several sines plotted on the same graphic.....	205
Figure 226: results of the script for plotting several sines.....	205
Figure 227: MATLAB logic and comparison operators	206
Figure 228: distribution of a series of points	207
Figure 229: interpolation of a series of data	208
Figure 230: graphic showing the interpolation of a series of data	208
Figure 231: script for solving a second degree equation	209
Figure 232: script for solving a second degree equation with complex roots	210
Figure 233: expanding and factoring an expression	210
Figure 234: script for deriving a function.....	211
Figure 235: script used to integrate a function and calculate a defined integral	212
Figure 236: script for calculating the Laplace transform of functions.....	212
Figure 237: script used to obtain the inverse Laplace transform of a function	213
Figure 238: script for partial fraction decomposition	215
Figure 239: script for solving a second order differential equation	217
Figure 240: solution 1 for the second order differential equation	219
Figure 241: solution 2 for the second order differential equation	219
Figure 242: transfer functions in series.....	221
Figure 243: transfer functions in parallel.....	222
Figure 244: closed loop transfer function.....	222
Figure 245: transfer functions of any system	223
Figure 246: step response of a system	224
Figure 247: impulse response of a system	224
Figure 248: Bode diagram of a system	225
Figure 249: Black-Nichols diagram of a system	225
Figure 250: Nyquist diagram of a system	226
Figure 251: two Bode diagrams plotted on the same graph.....	226
Figure 252: Bode diagram with indications of gain and phase margins.....	227
Figure 253: model of the servo control for the temperature of an oven	231
Figure 254: script containing simulation parameters	232
Figure 255: visualizing the variables created in the Workspace	232
Figure 256: response of the oven temperature.....	233
Figure 257: naming a Simulink signal.....	234
Figure 258: placing linearization points to plot an open loop Bode diagram	235

Figure 259: placing an “Open loop input” linearization point.....	235
Figure 260: view of linearization points.....	236
Figure 261: inserting a Bode Plot block.....	236
Figure 262: settings window for Bode Plot block.....	237
Figure 263: Bode diagram plot window.....	237
Figure 264: Bode diagram of the open loop.....	238
Figure 265: view of linearization points.....	239
Figure 266: inserting a second Bode Plot block.....	239
Figure 267: settings window for Bode Plot block.....	240
Figure 268: settings for the linearization points of the closed loop.....	240
Figure 269: Bode diagram plot window.....	241
Figure 270: closed loop Bode diagram.....	241
Figure 271: servo control for the temperature of an oven with saturation.....	242
Figure 272: response of the oven temperature with saturation.....	243
Figure 273: placing a scope over a signal.....	244
Figure 274: Displaying the output voltage from the saturator.....	245
Figure 275: configuration window for exporting data to the Workspace	247
Figure 276: script to plot a series of curves.....	248
Figure 277: influence of proportional correction on the oven temperature	249
Figure 278: course loop without state diagram.....	251
Figure 279: positioning a chart in a Simulink model.....	251
Figure 280: commands for creating a state and a “default transition”	252
Figure 281: creating system states.....	252
Figure 282: creating a default transition.....	253
Figure 283: creating transitions between states.....	253
Figure 284: attaching actions to states	254
Figure 285: writing transition labels in a states diagram	254
Figure 286: chart not connected with the model.....	255
Figure 287: adding variables to the diagram using Symbol Wizard.....	255
Figure 288: displaying variables in Model Explorer.....	256
Figure 289: chart before connection with the system.....	257
Figure 290: modifying the port numbers of a chart.....	257
Figure 291: connecting the chart with the Simulink block diagram.....	258
Figure 292: simulation results	258
Figure 293: example of superstate and substate.....	259
Figure 294: view of parallel and exclusive states	260
Figure 295: creating superstates.....	261
Figure 296: creating parallel substates.....	262
Figure 297: parallel states in Stateflow	262
Figure 298: using internal variables to evaluate the activation of a state	263
Figure 299: creating an output variable to Simulink.....	263
Figure 300: completed course control chart with superstates and parallel states	264
Figure 301: connect the chart output variable to a scope.....	264
Figure 302: displaying the activation state of the light.....	265
Figure 303: SimMechanics 2G model of the mechanical part of the hydraulic pilot.....	268
Figure 304: SimMechanics 2G model of the hydraulic pilot with masks over the solids.....	269
Figure 305: Mechanics Explorer window showing the mechanical section of the hydraulic pilot.....	269
Figure 306: SimMechanics visualization bar options	270
Figure 307: setting gravity in SimMechanics	270
Figure 308: modifying the action of gravity	271
Figure 309: hydraulic pilot model with SimMechanics model to be integrated	271
Figure 310: view of the SimMechanics system not connected to the rest of the model	272
Figure 311: placing the connection ports in the model.....	273
Figure 312: connecting the sub-system ports to the model.....	273
Figure 313: translational interface between Simscape and SimMechanics	274
Figure 314: rotational interface between Simscape and SimMechanics	274
Figure 315: interfaces between Simscape and SimMechanics	275
Figure 316: addition of Simscape SimMechanics Interface block	275
Figure 317: configuring the ports for a connection	276
Figure 318: Connection between Simscape and SimMechanics	277
Figure 319: adding a port to introduce torque to a connection.....	277
Figure 320: conversion of physical signal into a Simulink signal.....	278
Figure 321: adding a port to a connection in SimMechanics.....	278
Figure 322: viewing the offset of the helm angle	279
Figure 323: compensating for the offset of the helm angle	279
Figure 324 : hydraulic_pilot_SimMechanics_end_US.slx.....	280

Figure 325: adding a force to a connection, using a Lookup Table.....	281
Figure 326: configuring a Look-up Table.....	281
Figure 327: viewing a Lookup Table	282
Figure 328: view of the curve representing the input-output law for a Lookup Table	283
Figure 329: Results of the simulation of the course followed by the boat.....	283
Figure 330: Choosing the SimMechanics Link version.....	284
Figure 331: Solidworks welcome window	286
Figure 332: Opening the Solidworks add-on window	286
Figure 333: Selecting Solidworks add-ons	287
Figure 334: Opening the Solidworks model of the mechanical part of the pilot	288
Figure 335: Converting the assembly to an xml file.....	288
Figure 336: End of the file conversion.....	289
Figure 337: SimMechanics model obtained.....	289
Figure 338: Mechanics Explorer window	290
Figure 339: "Black box" modeling	291
Figure 340: Test conditions.....	292
Figure 341: Display of the data used for identification.....	292
Figure 342: Display of the data used for identification in Workspace.....	293
Figure 343: Description of the toolbox identification window	293
Figure 344: Importing time domain data	294
Figure 345: Selection of time domain data for identification.....	294
Figure 346: Display of imported data in the "System Identification" toolbox window.....	295
Figure 347: Display of imported data in the "System Identification" toolbox	295
Figure 348: Choosing the model type in the "System Identification" toolbox	296
Figure 349: Choosing the transfer function form in the "System Identification" toolbox.....	296
Figure 350: The Plant Identification Progress window of the "System Identification" toolbox.....	297
Figure 351: Displaying the identification results.....	298
Figure 352: Transfer function obtained by identification.....	298
Figure 353: Overlap of the data from the experiment and the model resulting from the identification	299
Figure 354: DC motor modeling	302
Figure 355: Speed control of a DC motor	303
Figure 356: Time domain response of the uncorrected system at a unitary level.....	303
Figure 357: Adding a PID controller block into the control system.....	304
Figure 358: PID Controller configuration window.....	304
Figure 359: Settings window for system performances	305
Figure 360: System performance criteria and compensator gain values.....	306
Figure 361: Adding a Bode diagram to the open-loop transfer function.....	306
Figure 362: Display of the Bode diagram of the open-loop transfer function for PID settings	307
Figure 363: Setting the performance criteria options	307
Figure 364: Viewing performance criteria	308
Figure 365: Automatically updating the PID parameters	309
Figure 366: System response after PID adjustments	309
Figure 367: Speed controlled DC motor model with disturbance	310
Figure 368: Time domain response of the uncorrected system	311
Figure 369: Placing linearization points on the model.....	312
Figure 370: Control and Estimation Tools Manager window	312
Figure 371: Viewing the input and output signals taken into account for the closed-loop.....	313
Figure 372: Choosing a block to adjust	313
Figure 373: Design Configuration Wizard window.....	314
Figure 374: Choosing plots to view to adjust the PID	315
Figure 375: Choosing closed-loop responses.....	316
Figure 376: LTI Viewer for SISO Design Task	317
Figure 377: SISO Design for SISO Design Task	318
Figure 378: Modification of Analysis Plots choices.....	319
Figure 379: Updating the Linear Analysis window for SISO Design Task.....	320
Figure 380: The Compensator Editor tab of the Control and Estimation Manager window	321
Figure 381: Functions of the PID adjustment window	322
Figure 382: Definition of speed criteria.....	323
Figure 383: Definition of performance criteria.....	323
Figure 384: Display of the performance criteria in the graphic window	324
Figure 385: Settings satisfying PID specifications	325
Figure 386 : Results of manual adjustment to the PID on the time domain response.....	326
Figure 387: Selecting the step response in accordance with the disturbance	327
Figure 388: Step response of the system in relation to a level of disturbance	327
Figure 389: Influence of adjustments on Design Plots	328
Figure 390: Confirmation of the importation of the parameters of the settings parameters in Simulink	329

Figure 391: Time domain response after adjusting the PID	329
Figure 392: DC motor control model with control by transfer function for adjustments.....	330
Figure 393: Non-corrected system response.....	331
Figure 394: Control and Estimation Tools Manager window	332
Figure 395: Choosing a block to adjust	332
Figure 396: Information window for the Control System Designer tool.....	333
Figure 397: Choosing the poles to view for the open-loop.....	333
Figure 398: Selecting display plots of the closed-loop performance.....	334
Figure 399: Display of closed-loop transfer function performance	335
Figure 400: The command bar of the SISO Design for SISO Design Task window.....	336
Figure 401: Gain variation of the open-loop transfer function	337
Figure 402: View of the influence on the closed-loop performance	338
Figure 403: Displaying the compensator transfer function	339
Figure 404: Adding an integrator	340
Figure 405: Visualization of the instability of the step response	341
Figure 406: Visualization of the poles and zeros on the curves.....	341
Figure 407: Stabilizing the system by adding a lead compensator	342
Figure 408: Closed-loop performance after lead compensation	343
Figure 409: Viewing the compensator transfer function	344
Figure 410: Adding a rejecter filter to the compensator	345
Figure 411: Influence of the notch on the performance of the closed-loop	345
Figure 412: Adjusting the notch	346
Figure 413: Performance of the closed-loop after adjusting the filter.....	347
Figure 414: Viewing the compensator transfer function	348
Figure 415: Influence of the filter on the behavior of the open-loop	349
Figure 416: Corrected step response	350
Figure 417: Comparison of corrected and non-corrected step responses	350
Figure 418: Scopes measuring the current in the circuit and the voltage across the coil terminals	351
Figure 419: Measuring the voltage across the coil terminals.....	351
Figure 420: Scope command bar	352
Figure 421: Formatting curves in the scope	353
Figure 422: Changing the format of a curve in the scope	353
Figure 423: Changing the number of inputs of a scope	354
Figure 424: Connecting one scope to two inputs	354
Figure 425: View of two curves in the same scope	355
Figure 426: Changing the characteristics of the second curve	355
Figure 427: Changing the characteristics of the second curve	356
Figure 428: Changing the number of windows in the scope	356
Figure 429: Displaying curves in a scope in multiple windows	357
Figure 430: Using cursors in a scope.....	358
Figure 431: Signal Builder settings window	361
Figure 432: Signal Builder command bar	362
Figure 433: Choosing the duration of the signal in "Signal Builder"	362
Figure 434: Manually moving segments in the "Signal Builder"	363
Figure 435: Moving one point in the "Signal Builder"	363
Figure 436: Placing new points on the signal	364
Figure 437: Formatting the signal by moving segments and points	364
Figure 438: Generating many signals	365
Figure 439: View of multiple outputs in the "Signal Builder"	365