POLITECNICO DI TORINO

Collegio di Ingegneria informatica, del cinema e meccatronica

Corso di Laurea Magistrale
in Ingegneria Meccatronica (Mechatronic Engineering)

Masters Degree Thesis

Development and test of an automated RaLa needle positioning-Synthesis

Academic Tutor

Prof. Luigi Mazza

Company Tutor

Dr. Ing. Heiko Baum

Candidate

Hassan Attieh

April 2019

Acknowledgment

Firstly, I would like to express my sincere gratitude to my advisors ***Prof. L. Mazza*** & ***Dr. Ing. H. Baum*** for the continuous support of my thesis study and related research, for their patience, motivation, and immense knowledge. Their guidance helped me in all the time of research and writing of this thesis.

I would like to acknowledge my colleagues from my internship at "***Fluidon gmbh***" for their wonderful collaboration as you supported me greatly and were always willing to help me. I would therefore like to thank: Mr. B. Erzberger, Mr. E. Pasquini, Mr. D. de Ben, Ms. Katja Juschka & Mr. O. Breuer for their valuable guidance. You provided me with the tools that I needed to choose the right direction and successfully complete my thesis.

Last but not the least; I would like to thank my parents for their wise counsel and sympathetic ear and my brother and sisters for supporting me spiritually throughout writing this thesis. You were always there for me.

Thank you all!!

Abstract

This thesis was done in the premises of *Fluidon gmbh,Aachen,Germany* under the supervision of the academic tutor *Professor Luigi Mazza* and the company tutor *Dr.Ing.Heiko Baum*.

The pressure pulsations generated by a pump are affected by reflections in the connected piping network, so that the temporal and spatial course of the pressure pulsations is specific to the respective installation situation. In order to be able to evaluate the pulsation behavior or the characteristic pulsation of a pump as generally as possible and independently of the installation situation, it is therefore necessary to eliminate the influence of the connected system on the pressure-time curve. The RaLa (*Reflexionsarmer Leitungsabschluss, Low-Reflection Line Termination*) is a device that can be adapted to the impedance of the pipeline so that a reflection-free state in the line can be achieved in order to obtain accurate results from the test-rig measurements done on the pump. It acts as an adjustable shutter with volume connected up downstream & inserted in the hydraulic system between the pump & the load. Its functionality depends on the matched impedance concept, where a flow restriction is installed in the pipeline system and by sizing it to the physical characteristics of the flow system we can guarantee that no incident pressure pulsations are to be reflected. The sizing of this flow restrictor, which is the RaLa in this case, is done through a needle (connected to a spindle) that moves towards or far from the shutter to adapt its impedance to the characteristic impedance of the pipeline (Ex: fuel delivery system) until a reflection-free state is achieved. The needle is adjusted & shifted to the desired position through a hand wheel where the user keeps track of the output signals through a scope displaying them on the screen. We have two 1-meter apart dynamic pressure sensors installed on the pipeline where we continuously acquire their signals and compare them. In case of matched impedance, the pressure transients at both sensors would look similar to each other when the time delay between them is eliminated as there would be no reflected waves to force this difference.

Adjusting the needle with the hand wheel is time consuming and results sometimes in inaccurate results as a 0.1 mm-shift in the needle can make a difference. Thus, an algorithm was developed to control the needle position automatically through a Faulhaber drive motor which, alongside with the rest of the sensors and electrical devices, communicates with the controller through the *Beckhoff* terminal box which contains a group of Digital and Analog terminals where this communication takes place via a real-time Ethernet network called *EtherCAT*. The basic idea of the algorithm is that it compares the signals coming from the dynamic pressure sensors after compensating the time-delay between the signals and subtracting the static pressure value of the tank to eliminate the vertical offset. The controller takes the peak and the x-intersection of every oscillation from both signals, subtracts them and minimizes the difference through a specific algorithm. The hydraulic model was implemented in the DSHplus software while the controller algorithm was developed in Simulink and the co-simulation method was used to test the latter on the hydraulic model.

Table of Contents

1 RaLa

1.1 Introduction

Fluid-borne noise is a major contributor to air-borne noise and structure-borne noise in hydraulic systems leading to increased fatigue in its components. One of the significant contributors to this noise phenomenon in hydraulic systems is the set of pressure pulsations generated by the hydraulic positive-displacement pumps, as this is the class of pumps that is going to be used in the pump model, during the development of the controller algorithm & in the simulations.

The evaluation of the characteristic pulsation of a hydraulic positive displacement pump is of great significance for the design of the pump & its optimization. The flow of the pressure pulsations, especially due to the sharp rise of pressure in the pipeline, contributes in the rapid increase of the pressure ripples which leads to the increase of the reflected pressure waves at the end of the pipeline. The pump geometry plays a major role too which leads eventually to the increase of the fluid-borne noise that has the ability to exert strain on the pipeline system; such effects have undesirable and serious impact on the pipeline and the systems downstream.

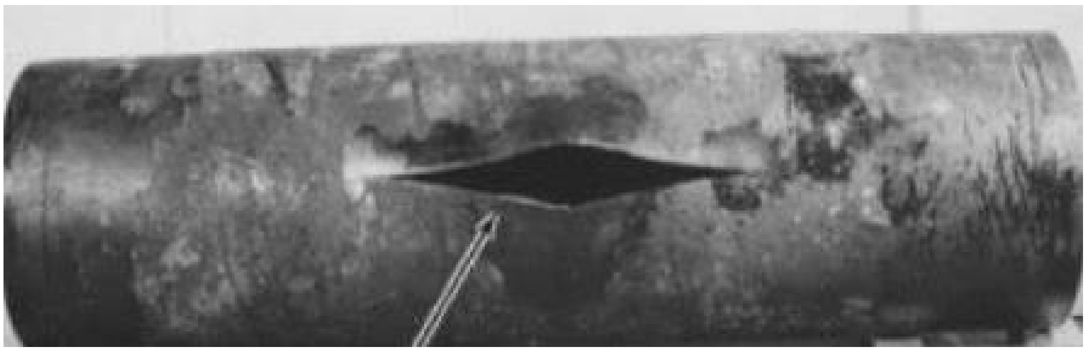


Figure 1: Fatigue cracks in the pipeline

Hydraulic positive displacement pumps are widely used in the automotive sector, especially in the fuel injection systems. An important requirement in any automobile fuel supply line is the smoothness of the flow of the fluid which can be achieved with a proper design of the fuel injection pump. Computer Simulations & test rig validations combined together form a powerful tool to provide users with practical feedback when designing real world systems. This allows the designer to determine the correctness and efficiency of a design before the system is actually constructed.

Figure 2: EA888 gen3 high pressure fuel pump (positive displacement pump)

In order to obtain accurate measurements on the test-rig that match the results obtained with the computer simulations, the pressure pulsations used in the measurements should belong only to the original waves generated by the pump and no reflected pressure waves are to interfere with the measurements.

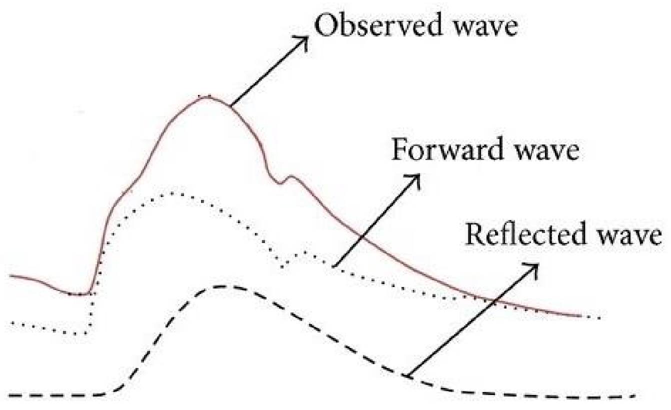


Figure 3: The interference of the reflected wave with the forward wave [1]

RaLa, in few words, is a low reflection line measurement device developed by "FLUIDON gmbh" which is a German company based in Aachen that specializes in performing computer simulations of hydraulic systems using mainly DSHplus which is a software they created too.

The name "RaLa" is an abbreviation of the German term (Reflexionsarmer Leitungsabschluss). It acts as an adjustable shutter with volume connected up downstream & inserted in the hydraulic system between the pump & the load. Its functionality depends on the matched impedance concept, where a flow restriction is installed in the pipeline system and by sizing it to the physical characteristics of the flow system we can guarantee that no incident pressure pulsations are to be reflected. The sizing of this flow restrictor, which is the RaLa in this case, is done through a needle (connected to a spindle) that moves towards or far from the shutter to adapt its impedance to the characteristic impedance of the pipeline (Ex: fuel delivery system) until a reflection-free state is achieved.

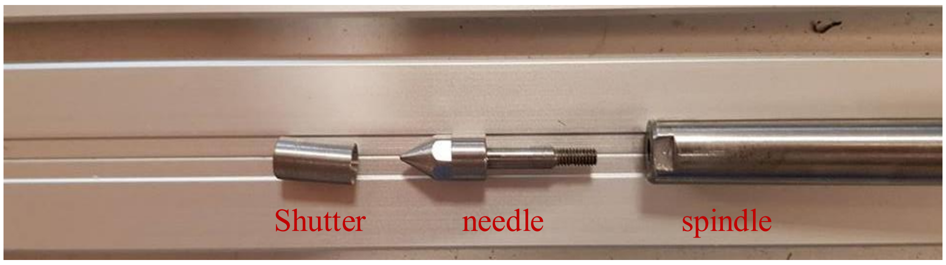


Figure 4: The RaLa needle, spindle & shutter

1.2 Needle Factor and Matched Impedance

"The matched impedance concept can be described as an installation of a flow restriction in a line so that pressure pulses arriving at the restriction are not reflected. This is accomplished by sizing the restriction to the physical characteristics of the flow system, including the fuel properties.

Consider a transient which is a simple step increase in pressure that passes through an originally steady flow system, in which losses (except for the flow restrictor) can be neglected. This pressure rise, ΔP, results in an increase in volumetric flow rate, ΔQ, of

$$\Delta Q = \frac{A\Delta P}{\rho\, a}$$

According to basic water hammer analyses, in which A = pipeline cross-sectional area, $\boldsymbol{\rho}$ = fluid mass density, and a = wave propagation velocity. The quantity Z = $\boldsymbol{\rho}$ a/A is the characteristic impedance of the pipe. This pressure pulse propagates through the system until it arrives at the flow restrictor where it will be modified." [2]

The final equation that calculates the pressure difference at the RaLa and is used by DSHplus to simulate its real behavior is:

$$\Delta P = \frac{\sqrt{\beta * \rho}}{A} * \frac{Q * needle\ factor}{2} \qquad \textbf{(Equation 1)}$$

Q=volumetric flow rate which is considered unitary as it doesn't make a difference whatever value it holds.

β=bulk modulus=11526.1 bar in our case.

$\boldsymbol{\rho}$=fluid mass density=756.4 Kg/m^3 in our case too.

A $[m^2]$=pipeline cross-sectional area=$\frac{(diameter)^2}{4\pi}$

In order to obtain matched impedance between the characteristic impedance of the pipeline and the impedance of the flow restrictor (RaLa), both impedances have to be equal:

$Z_{flow\ restrictor} = Z_{pipe}$ **(Equation 2)**

We know that:

$Z_{pipe} = \frac{\rho\ a}{A}$ **(Equation 3)**

For orifices:

Q=A*Cd*$\sqrt{\frac{2\Delta P}{\rho}}$ **(Equation 4)**

Where Cd=discharge coefficient which depends on the flow rate. So this equation can be represented in another form:

Q=B*ΔP^n **(Equation 5)**

In the worst case, which is a fully turbulent flow, the exponent "*n*" is equal to 2 while in a laminar case it is unity. Rearranging "**Equation 5"** we obtain:

$\Delta P = (\frac{Q}{B})^{1/n}$ **(Equation 6)**

Where "**B**" is only proportionality constant or a dimensionless quantity that is going to be eliminated later. Also in this case: 0.5<n<1.

$Z_{flow\ restrictor} = \frac{\partial \Delta \mathrm{P}}{\partial \mathrm{Q}}$ **(Equation 7)**

By combining "**Equation 5**" and "**Equation 7**" together we obtain:

$Z_{flow\ restrictor} = (\frac{1}{n})\ (\frac{1}{B})\ (\frac{Q}{B})^{\frac{1}{n}-1}$ **(Equation 8)**

According to "**Equation 5**":

B=$\frac{Q}{\Delta P^n}$ **(Equation 9)**

By combining "**Equation 8**" and "**Equation 9**" together we obtain:

$Z_{flow\ restrictor} = \frac{\Delta \mathrm{P}}{n\mathrm{Q}}$ **(Equation 10)**

We obtain the final formula by substituting the characteristic impedances obtained from "**Equation 3**" and "**Equation 10**" and knowing that: $\boldsymbol{\rho}$a=$\sqrt{\rho\beta}$:

$$\Delta P=\frac{\sqrt{\beta*\rho}}{A}*Q*n \qquad \textbf{(Equation 11)}$$

In the DSHplus model the maximum input value of the RaLa is unity, but in case we want to simulate the case where the shutter is wide open (i.e. greater than 1) , we divide this needle factor by 2 and we obtain a new limit: 0 < needle factor < 2 where 0 is when it is fully closed.

2 Hybrid Pump Model for 1D Hydraulic System Simulation

Goal: Finding a proper modeling technique to incorporate the dynamic behavior of a hydraulic displacement pump into the time based simulation.

Summary: The hybrid pump model is to be considered as a well-qualified model to represent the dynamic behavior of a positive displacement pump in the simulations that we are going to need when the controller algorithm is developed. This model was developed by engineers: Dr. Ing. Heiko Baum, Ing. Klaus Becker and Ing. Axel Faßbender, so this chapter was paraphrased from their published paper. [3]

2.1 Introduction

Fluid-borne noise in hydraulic systems mainly stems from the commonly used positive displacement pumps. Due to their discontinuous working principle, providing a flow rate through these types of pumps is accompanied by the generation of flow ripple. The flow ripple in turn leads to pressure oscillations, commonly referred to as noise. This noise has a great impact while examining the pressure oscillation phenomena or the noise creation in these systems.

In the latter case, it is required to provide a detailed physical representation of the pump in order to accurately represent its dynamic behavior, for instance: the pump impedance & the flow pulsations. While the rest of the hydraulic components like the pipes & the valves can be easily inserted in the model for further simulation in the time-domain, the process to obtain an elaborate pump model is tedious and costly. Due to the previously mentioned reasons, the need of a simple hybrid model that accurately represents the dynamic behavior of the pump arose. The systematic approach that was followed to create a hybrid pump model is to be elaborated in the following subchapters taking an automotive power steering vane pump as an example for the necessary simulations & validations.

2.2 Norton's Model

This model is substantially used by researchers especially in the frequency domain simulations.

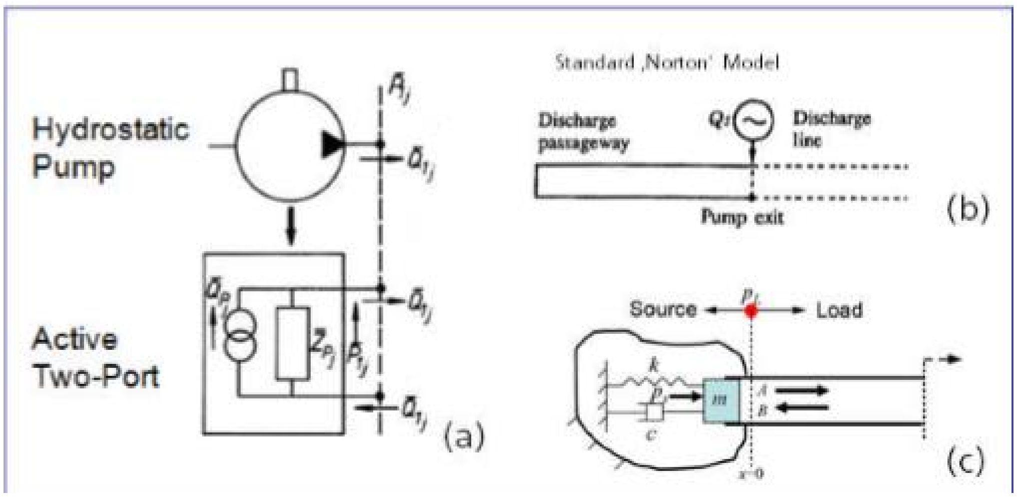


Figure 5: Pump representation as active dual pole and equivalent analogies (a) Stulemeijer,(b) Kojima, and (c)Liu

The 'Norton' model (b) is a simple representation of an active two-port (a), where the pump's discharge passageway is modeled as a pipe. The model is usually adjusted to a specific frequency which is the measured pump impedance, but a major drawback to this approach is the inaccuracy of the results at high frequencies. In order to avoid this outcome, the pump's discharge passageway is to be considered as a Helmholtz resonator or as a spring-mass damper system, but even in this case, the parameters of these first order oscillators have to be adjusted which doesn't reflect the real physical parameters of the pump.

2.3 The concept of the pump model

The 'Norton model' will form the basis of our model but it is necessary to develop its current form by introducing some notions from the wave decomposition approach such as the one presented by *Liu* and *Herrin*.

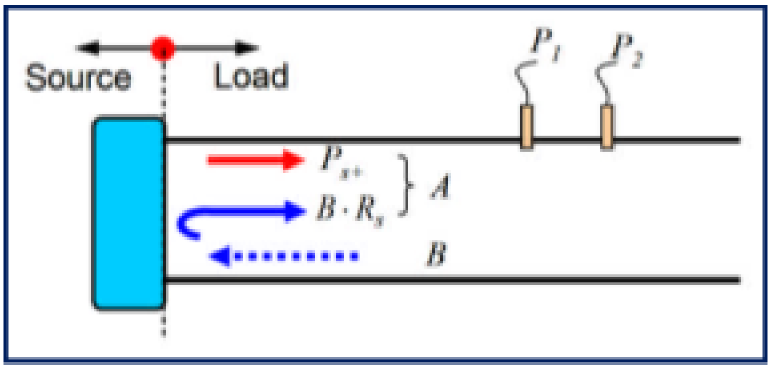


Figure 6: Schematic illustration of the wave decomposition theory

The wave decomposition approach states that the pump flange pressure p_L is a superposition of the pump's output pressure wave A and the system's reflected wave B. The output pressure wave A is also a superposition of the pump's internal pressure p_{s*} and the reflection of wave B at the pump's inner borderline, which can be expressed as B$\cdot R_s$ where R_s is the pump's reflection factor. The following equation expresses the previous notions mathematically:

$p_L = p_{s*} + B \cdot (1 + R_s)$ **Equation (12)**

In the ideal case, the RaLa is going to eliminate any reflected pressure oscillations which are represented in this case by the reflected wave B that is going to be zero in this case. Thus, p_L becomes equal to p_{s*}. Knowing that the discharge passageway pipe of our 'Norton' model shouldn't affect the shape of the pressure pulsation, it must have the same load impedance, consequently, the same diameter of the pipe used in the RaLa measurements. As a result, the pipe's length l_{RP} must be adjusted according to **Equation (13)** as a function of the measure pump's anti-resonance impedance f_I:

$l_{RP} = \frac{c_0}{4 \cdot f_I}$ **Equation (13)**

In order to represent the dynamic feedback B$\cdot R_s$ of the discharge passageway pipe & include it in the simulation model, a resistance is inserted into the resonance pipe

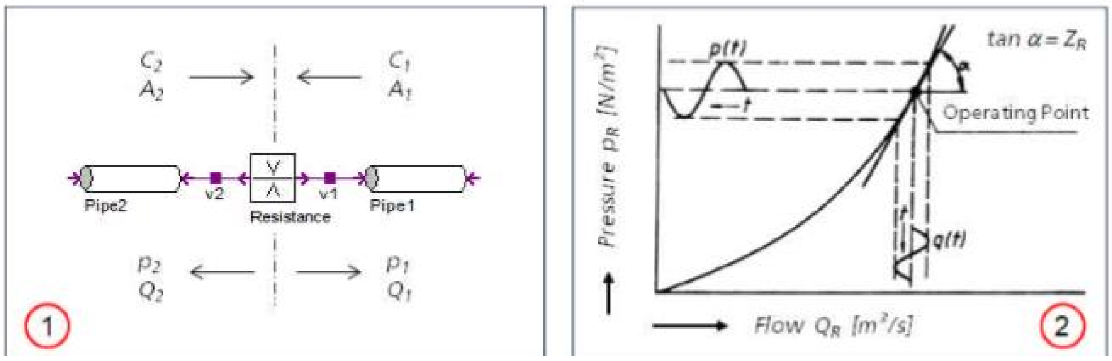


Figure 7: Resonance tube with orifice type resistance (1) Example of a distributed parameter model; (2) Impedance of the orifice's operating point.

As shown in part (1) of **Figure 7**, an orifice is installed between the two pipes in order to take the reflection factor into consideration in the time-based simulation.

The resulting impedance Z_R of the orifice:

$$Z_R = \frac{p_R}{Q_R} \qquad \textbf{Equation (14)}$$

The pressure-flow curve that can be seen in the second part of **Figure 7** has a non-linear characteristic so Z_R is to be calculated for every operation point, but by calculating the value of the flow through the orifice Q_R at each point we can automatically obtain Z_R during the simulation.

$$Q_R = \frac{C_2 \cdot Z_2 - C_1 \cdot Z_1}{Z_R + Z_2 + Z_1} \qquad \textbf{Equation (15)}$$

Z_2 & Z_1 are the impedances of the connecting pipes [Pa · s/m)] while C_2 & C_1 represent the wave characteristics of these pipes

$$p_1 = (Q_{S*} - Q_R - C_1) \cdot Z_1 \qquad \textbf{Equation (16)}$$

$$Q_1 = (Q_{S*} - Q_R - (C_1 + \frac{p_1}{Z_1}) \qquad \textbf{Equation (17)}$$

2.4 Measurements

Two test rigs are introduced & the measurement data is acquired from both of them. The first one is used to measure the pump's characteristic pressure pulsation when a RaLa device is inserted in the circuit while the second test rig measures the pump's impedance using the 2p/2s approach.

2.4.1 RaLa

The test rig is set up as following:

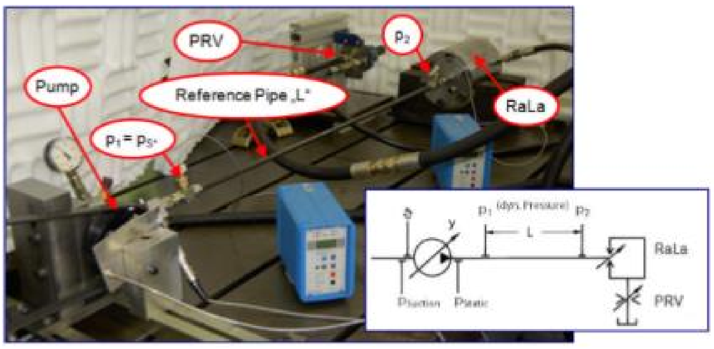


Figure 8: The setup of the RaLa test-rig

The measurement of the pressure oscillation p_{S*} is converted into a flow pulsation Q_{S*} by the following equation:

$$Q_{S*} = \frac{A_{RP}}{\rho \cdot c_0} \cdot p_{S*} \qquad \textbf{Equation (18)}$$

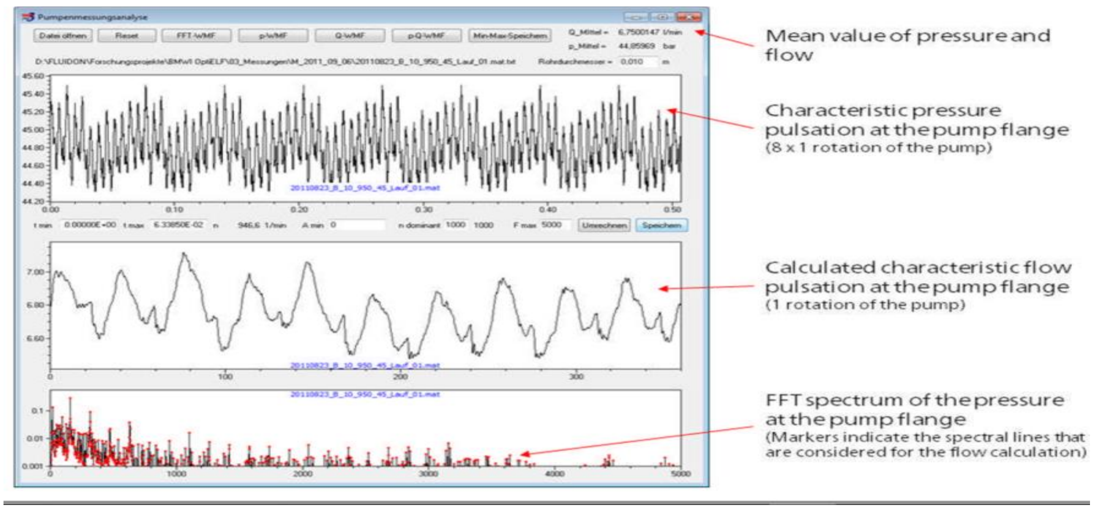

Pumpenmessungsanalyse
Datei öffnen
Reset
FFT-WMF
p-WMF
Q-WMF
pQ-WMF
Min-Max-Speichern
Q_Mittel = 6,7500147 l/min
p_Mittel = 44,85969 bar
Mean value of pressure and flow
Characteristic pressure pulsation at the pump flange
(8 x 1 rotation of the pump)
Calculated characteristic flow pulsation at the pump flange
(1 rotation of the pump)
FFT spectrum of the pressure at the pump flange
(Markers indicate the spectral lines that are considered for the flow calculation)

Figure 9: Conversion of the pressure pulsation into a flow pulsation

Using a constant sampling rate decreases the number of data samples as the pump speed increases.

The data is stored in look-up tables, the y-axis consists of the volumetric flow rate values as a function of the pump's angular position values varying from 0° to 360° (x-axis).

2.4.2 2p/2s

The test rig is set up as following:

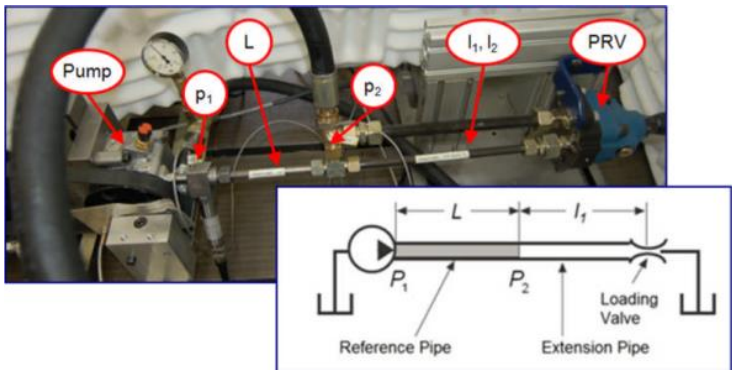


Figure 10: The setup of the 2p/2s test-rig

The pump impedance Z_s is:

$$Z_s = jZ_c \frac{(P_1 - P_1') \sin(\beta \cdot L)}{P_2 - P_2' - (P_1 - P_1') \cos(\beta \cdot L)}$$ **Equation (19)**

The length of the λ/4 resonance pipe of the hybrid pump model is obtained from the calculated pump impedance which by its turn will be used to verify the final hybrid pump model parameterization by comparing it to the obtained pump impedance from the simulations.

2.5 The setup of the hybrid pump model

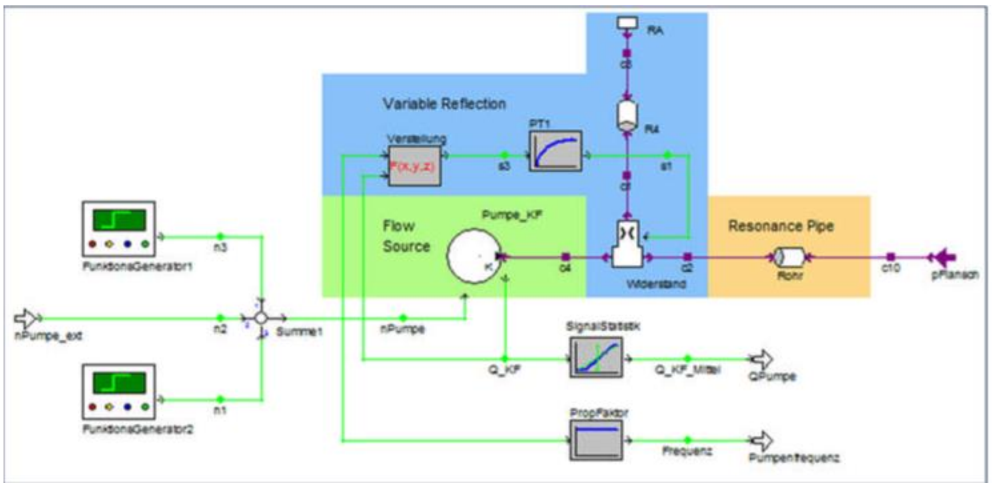


Figure 11: Simulation model representation of the pump

The system’s input is the pump’s rotational speed & the system pressure is the third set of data that is to be inserted in the look-up table alongside with the rotational angle & the volumetric flow rate.

It is important to state the following facts:

-The pump’s rotational speed & its volumetric flow rate control the orifice’s impedance & accordingly the reflection coefficient of the source.

-As mentioned in the beginning, in the ideal case, no pressure oscillations are going to be reflected in the presence of the RaLa, thus having the diameter of the λ/4 resonance pipe equal to the one of the reference pipe used in the RaLa measurement.

-For the future uses in other simulation models as the one used in this thesis, this model is to be considered as an enclosed subsystem within the whole model as it can be seen in the following figure.

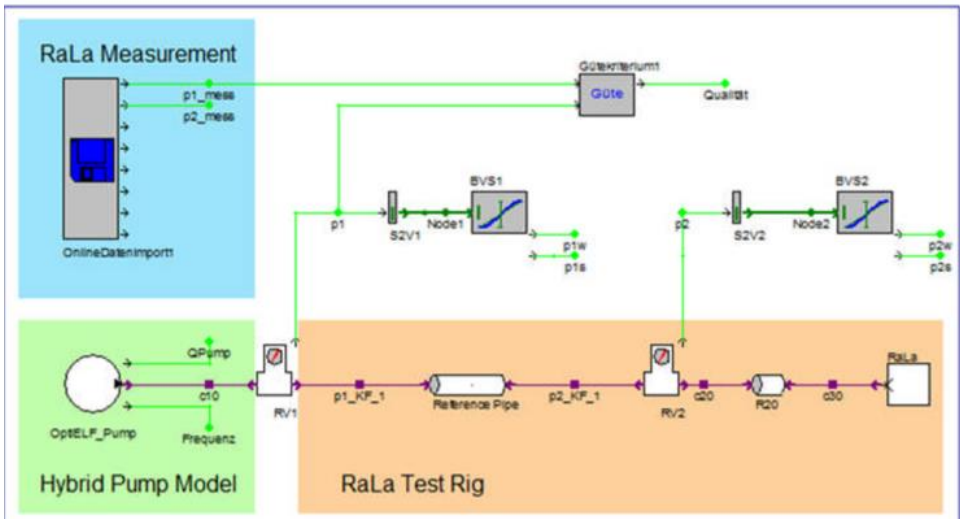


Figure 12: Encapsulation of the sub-modules in the DSHplus model

3 Computer simulation software & DSHplus model

Summary: Computer simulations use a mathematical description or model of a real system to simulate the dynamic behavior of its components which saves time & resources. In the following headlines, we're going to discuss the definition, pros and cons of every method, concluding that performing the simulations on the computer is more advantageous than doing the initial tests and simulations on a physical prototype or a real test-rig. This will form an introduction to the importance of the DSHplus software; its contribution to this thesis and after the DSHplus model of the project is going to be discussed in detail.

3.1 Introduction

Computer simulations mainly study the behavior of a system without building it which makes it an "easy to perform" process because it uses the "what-if" analysis. Through the use of this method, an engineer will be able to deal with large amounts of data easily as compared to a case whereby the simulation wasn't being done with a computer. Therefore, different conditions & scenarios can be tested & the outcome can be investigated in a short process with respect to the analytical simulation that can be long, tedious and a waste of resources in most cases. DSHplus is powerful software that offers to the user an integrated CAE solution in fluid technology through which he can take advantage of the various libraries, interfaces pre-processing and post-process methods integrated into the software to master the simulations making them more realistic & similar to the results obtained on the test-rig.

3.2 Virtual Modelling (simulations) vs. Physical prototyping

3.2.1 Physical Prototyping

A prototype in general is an initial display of a design concept, either a scale or full-size model of a structure or piece of equipment, which can be used to evaluate performance, behaviour, or the interaction with other components in a physical system. Prototyping is the process of generating prototypes which provide the engineers with the opportunity to determine if a concept is technically feasible, optimize performance, understand interfaces between subsystems, or identify potential issues. A physical prototype is a preliminary representation of this concept in a tangible model.

The pros of physical prototyping are as follows:

- Complete & reliable: The physical prototypes fully represent a scale or a full-size model of the design concept.
- Can be used for tests where the prototypes can be inserted in any system & tested for its behavior & interaction with the other components.
- Provide a quick feedback, where the results can be seen in real-time.

While the cons are:

- It requires a long production time: one could potentially invest the time in other innovative research.
- Costly both moneywise & resource- wise.
- Difficult changes & alternatives: as any change in the parameters or in the design requires the generation of a new prototype which proves the second drawback by wasting both money & resources.

3.2.2 Virtual modelling

Virtual prototypes are digital mock-ups (computer simulations or analytical models) of a physical model that can be analyzed and tested in order to study the behavior of it & its interaction with the other components in a physical system. Practical examples of virtual modelling vary from the FEM (finite element analysis), CFD (computer fluid dynamics) & 3D CAD models (computer aided design).

The pros of virtual modelling are as follows:

- Low cost: this technology doesn't require generating any physical prototype as it requires only an operating system & proper simulation software like DSHplus.
- Easy to get organized information and acquire the simulation results in order to analyse them or even use this data as a database for other models.
- Ability to change the parameters of the design or change the design concept itself easily & test it in only the time required to insert the new component in the physical model.
- Saves time & resources.

The major drawback of virtual modelling, computer simulations specifically, is that sometimes an event that may occur instantaneously in the real world with the physical prototype may take several hours to mimic in the simulated environment. This depends on the number of components in the system, the complexity of the simulation (which may depend on the number of entities too) and the processing power invested in the simulation platform.

3.3 DSHplus: the simulation software & its contribution to our model

3.3.1 What is DSHplus?

"DSHplus is a simulation environment specialized on the dynamic non-linear calculation of complex hydraulic and pneumatic systems and components. In particular DSHplus is used for verification of system function, for analysis of the system dynamics, for system revisions, for component selection and design, in the fault diagnosis and for training purposes." (4)

As mentioned in the introduction, it is powerful software that offers to the user an integrated CAE solution in fluid technology through which he can take advantage of the various libraries, interfaces pre-processing and post-process methods integrated into the software to master the simulations making them more realistic & similar to the results obtained on the test-rig.

3.3.2 Co-simulation

An important property of DSHplus that contributes to rendering the simulations more realistic is the ability to export the hydraulic system model into a Simulink model and an S-function (MEX file) is generated. S-functions are dynamically linked subroutines that the MATLAB engine can automatically load and execute; they are simply a computer language description of a Simulink block.

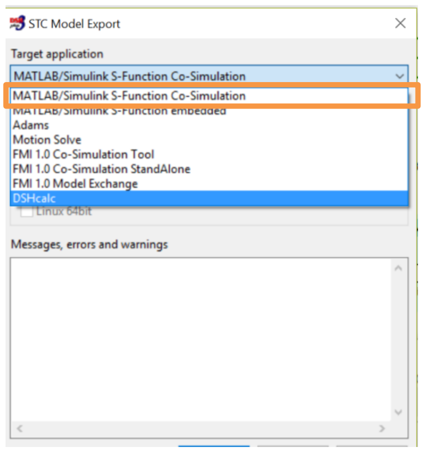

STC Model Export
Target application
MATLAB/Simulink S-Function Co-Simulation
MATLAB/Simulink S-Function Co-Simulation
MATLAB/Simulink S-Function embedded
Adams
Motion Solve
FMI 1.0 Co-Simulation Tool
FMI 1.0 Co-Simulation StandAlone
FMI 1.0 Model Exchange
DSHcalc
Linux 64bit
Messages, errors and warnings

Figure 13: Simulink model export in DSHplus

What is Co-simulation?

In simple words, co-simulation is an enabling technique that allows the models between two computer programs which are considered as simulation environments (Simulink and DSHplus in our case) to interact with each other by executing them simultaneously, allowing information and data to be shared between them at each time step.

In this thesis, the co-simulation was enabled through an S-function that was implemented as a co-simulation gateway between Simulink and DSHplus (**Figure 26**, p: 24). Simulink acts as a master while DSHplus acts a slave so DSHplus has to be connected to Simulink through the TCP/IP port by clicking on the "connect" button in DSHplus for the Simulink simulations to run. In conclusion, DSHplus has to be running for the co-simulation to work too.

3.3.3 RaLa hydraulic DSHplus model

In the following subdivisions, the functionality of various parts of the RaLa hydraulic circuit is going to be described while highlighting on the significance of some components and their contribution to make the simulations as realistic as possible.

The hydraulic model in DSHplus is used to emulate the behaviour of the incorporated components of the test-rig for the sake of developing a controller algorithm to control the needle. As a result, the DSHplus block in the Simulink model which communicates in a continuous matter at each time step with the hydraulic components through the corresponding port, is going to be replaced in the following steps with Etherlab blocks.

a) System Pressure and the needle factor

- **Needle Factor:**

The RaLa's needle factor, as previously discussed in detail, is basically considered similar to a screw or a hand wheel that tunes the aperture of the adjustable shutter in order to obtain matched impedance between the RaLa and the corresponding pipeline.

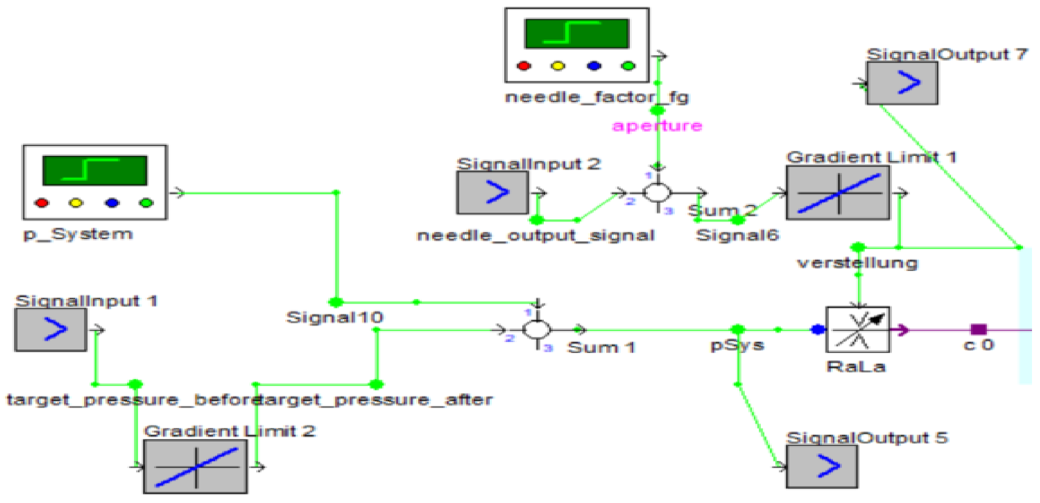


Figure 14: RaLa's needle factor and the system's target pressure in DSHplus

The needle factor as seen in **"Figure"** is fed to the RaLa block either through:

- The function generator *"needle_factor_fg"* which generates a signal of type "constant" and the value is inserted in DSHplus not in Simulink.

  <u>OR</u>

- The input module *"needle_output_signal"* which passes the value received from Simulink to the RaLa block.

The main functionality of the component *"Gradient Limit"* is to limit the gradient of the input signal according to the defined minimal and maximal gradients. We will exploit this property in the software to render the simulations more realistic by taking into consideration the presence of some delay in the signal due to the physical characteristics of the system, so this block is inserted between the output of the function generator/Simulink input module and the input of the RaLa block in order to deliver the received signal after some time according to the inserted values of the "gradient rising" and "gradient falling" parameters.

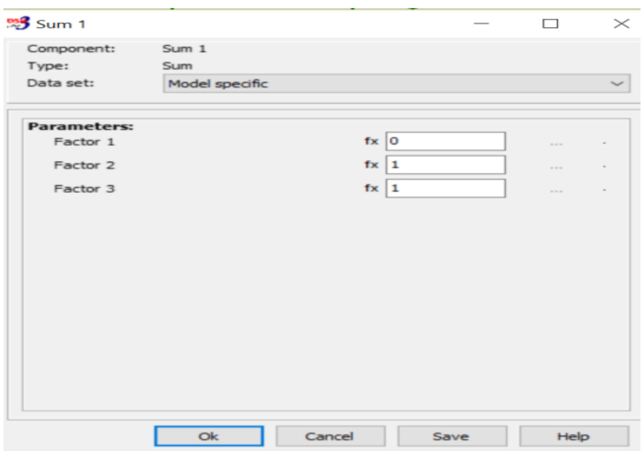
- **The system's target pressure:**

The system's target pressure as seen in **"Figure 14"** is fed to the RaLa block too either through:

- The function generator *"p_System"* which generates a signal of type "constant" and the value is inserted in DSHplus not in Simulink.

  <u>OR</u>
- The input module *"SignalInput 1"* which passes the value received from Simulink to the RaLa block.

The output module *"SignalOutput 5"* is supposed to support us with the current system pressure & send it to our Simulink model, where the controller algorithm will exploit this measurement to compare it with the value of "target pressure" inserted by the user insure that it's within a specific tolerance limit.



Component: Sum 1
Type: Sum
Data set: Model specific

Parameters:

Factor 1	fx	0
Factor 2	fx	1
Factor 3	fx	1

Figure 15: The *"Sum"* Block

Figure 15 shows the *"Sum"* Block which is found in both of the implementations:

i. Needle Factor
ii. The system's target pressure

This block is used to make sure that the only input signal is the one arriving from the chosen source (Simulink/DSHplus) by choosing the factor "0" to block the unwanted input signal and "1" for the input signal that is going to supply us with the required data which can be a constant value, waveform or a pulse signal.

b) Fluid temperature

This part is separate from the other components of the DSHplus circuit that contribute in a direct way in the algorithm. This part of the model simulates the temperature of the fluid rendering the simulation more realistic. **(Figure 16)**

There is only one input that comes from one of the two sources:

- The function generator *"Function Generator 2"* which generates a signal of type "constant" and the value is inserted in DSHplus not in Simulink. **(Figure 16)**

  OR

- The input module *"Temperature_input"* which passes the value received from Simulink to the DSHplus model.

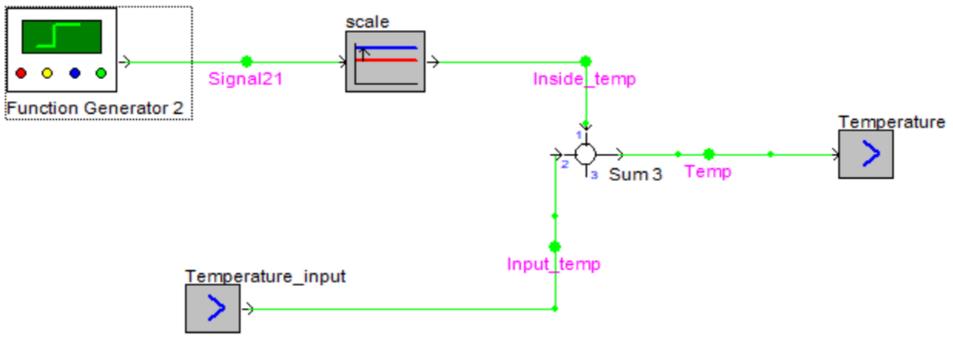
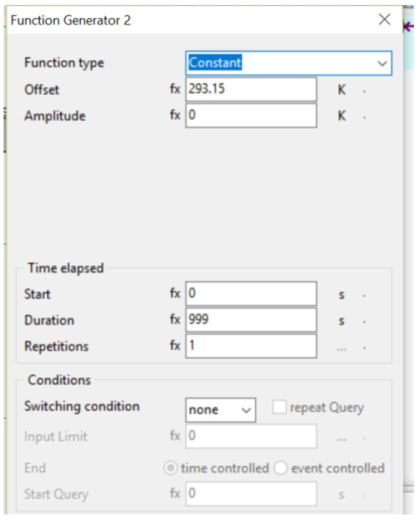


Figure 16: Fluid Temperature module blocks

The scale provides an offset of "-273.15" and is considered only for the function generator because the measuring unit of its output is "Kelvin (K)" while we want it in "Celsius (C)". The output is then passed to the Simulink model in order to display it and use it in specific connections and signals.



Function Generator 2

Function type		Constant	
Offset	fx	293.15	K
Amplitude	fx	0	K
Time elapsed			
Start	fx	0	s
Duration	fx	999	s
Repetitions	fx	1	...
Conditions			
Switching condition		none	repeat Query
Input Limit	fx	0	...
End		time controlled / event controlled	
Start Query	fx	0	s

Figure 17: The parameters of the temperature function generator block

c) Pressure pulsations

As seen in **Figure 18,** the components that interact with each other on order to provide us with the required data of the pressure pulsations are:

- *Sensor p1:* that provides us with the P1 dynamic pressure measurements.
- *Sensor p2:* that provides us with the P2 dynamic pressure measurements.

- *Konstante 1:* It is a constant value that represents the system pressure and we can change its value by changing the value of "systemDruck" from the global parameters section in DSHplus.
- *Totzeit 1:* It is used to compensate the time delay between the 2 signals: p1_dyn and p2_dyn.
- *"Summe 1" and "Summe 2":* They subtract the system pressure from the pressure measurements coming from the sensors: *Sensor p1* and *Sensor p2*.
- *"SignalOutput 1" and "SignalOutput 2":* They are responsible of passing the measurement data to Simulink.

This segment of the model is one of the most essential parts that contribute significantly in the algorithm as its data provides one of the few inputs of the controller's algorithm. This means that a close look has to be taken on every parameter inside these blocks because any minor change can affect the outcome and result in colossal damage to the output of the algorithm. One of the most important details that have to be taken care of is the units, as any shift in the measurement units will change the value of the pressure pulsations at each instance which will end up with misleading results. As you can notice, the *"p1dyn"* signal was sent to Simulink before compensating the time delay as it was important to deal with this part there because the DSHplus model serves solely as a realistic emulation of the test-rig's behavior.

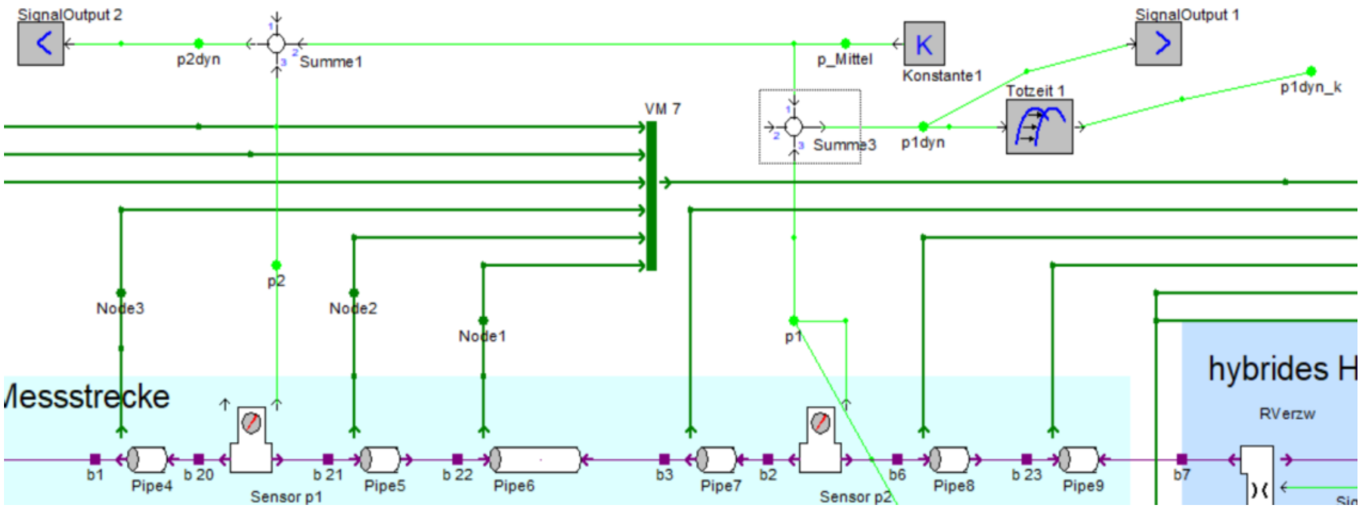


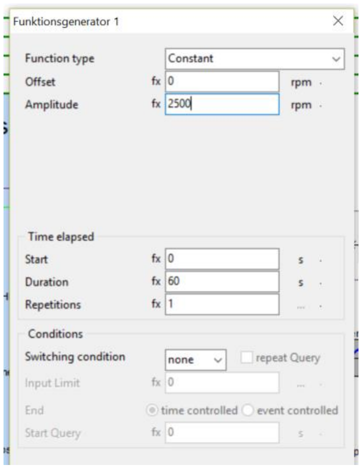
Figure 18: The pressure pulsations blocks

d) The pump model

The only components that are going to have the lights shed on in this part are the ones that contribute in the controller algorithm and that participate in the process of data exchange between DSHplus and Simulink.

The components are:

- *Funktionsgenerator 1:* This is the function generator which provides the pump block with the rpm value of the electric motor. As you can see in **Figure 19,** the type of the signal is a “constant” which is equal to the value of the desired rotational speed.

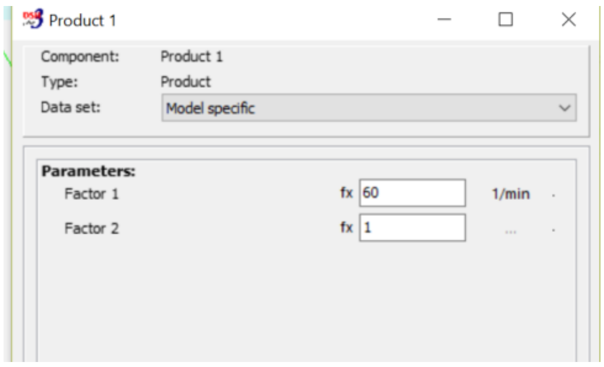


Funktionsgenerator 1

Function type		Constant	
Offset	fx	0	rpm
Amplitude	fx	2500	rpm
Time elapsed			
Start	fx	0	s
Duration	fx	60	s
Repetitions	fx	1	...
Conditions			
Switching condition		none	☐ repeat Query
Input Limit	fx	0	...
End		[✓ time controlled] event controlled	
Start Query	fx	0	s

Figure 19: The parameters used in “Funktionsgenerator 1” block

- *SignalInput 3:* This provides the rpm value of the electric motor from the Simulink model, so this value can be supplied by one of these two sources, either by the DSHplus function generator or the associated block in Simulink.
- *Gradient Limit 3:* As mentioned before, this causes a delay in the signal for more realistic simulations, as the electric motor’s speed is going to need some time to ramp up and can’t happen instantly like in DSHplus.
- *SignalOutput 6:* It passes the value to the Simulink model as it is going to be used in the controller algorithm.
- The far down part of **Figure 22** is used to emulate the behavior of the incremental encoder that measures the rotational speed of the electrical motor and the angle by taking 512 pulses / revolution.
- *Constant 1:* It holds the value of “512”.
- *Product 1:* It is used to multiply the rotational speed by the number of pulses per revolution to obtain the total number of pulses. As you can see in **Figure 20** the rotational speed was divided by 60 in order to change the unit from rpm to rps.



Product 1

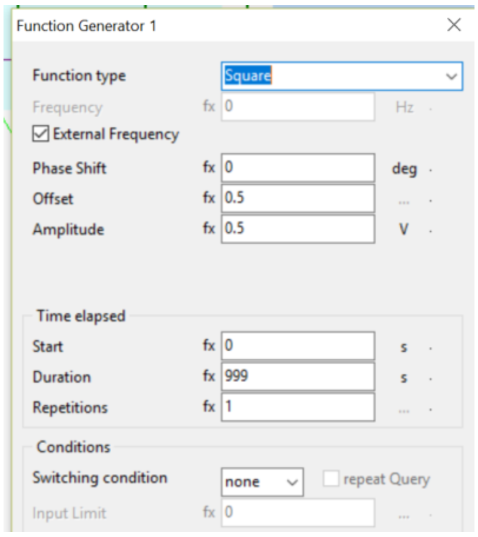
Component:	Product 1
Type:	Product
Data set:	Model specific

Parameters:

Factor 1	fx	60	1/min
Factor 2	fx	1	...

Figure 20: The parameters of the "Product 1" block

- *Function Generator 1:* It generates a square wave of frequency equal to the output of the product block. **(Figure 22)**



Function Generator 1

Function type		Square	
Frequency	fx	0	Hz
☑ External Frequency			
Phase Shift	fx	0	deg
Offset	fx	0.5	...
Amplitude	fx	0.5	V

Time elapsed

Start	fx	0	s
Duration	fx	999	s
Repetitions	fx	1	...

Conditions

Switching condition		none	☐ repeat Query
Input Limit	fx	0	...

Figure 21: The parameters of the "Function Generator 1" block

- *SignalOutput 4:* It passes the data to the Simulink model.

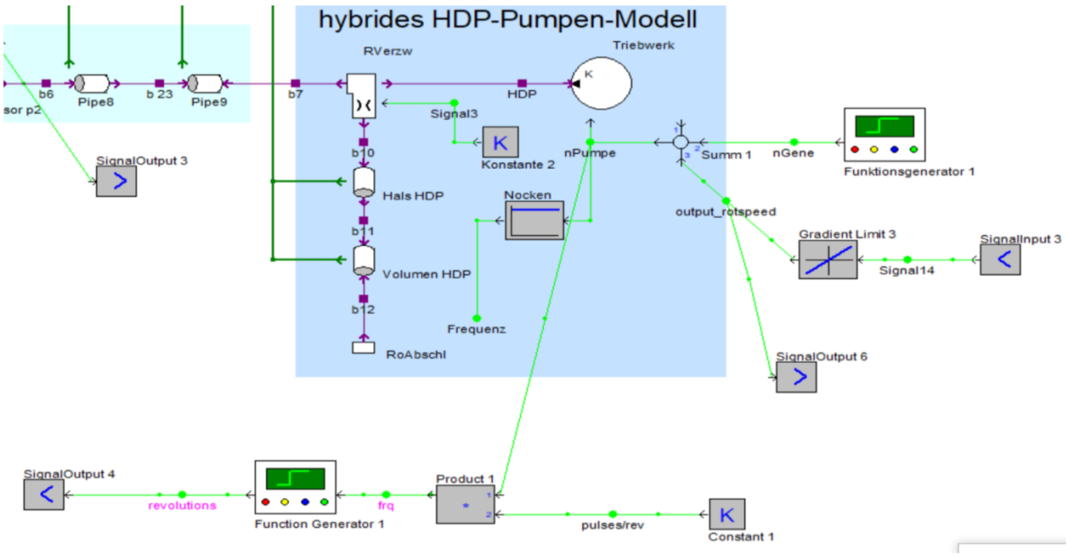


Figure 22: The pump model blocks

Reminder: The DSHplus block in the Simulink model presents a platform to exchange data between the models of the two programs but this block is going to be deleted later and changed with some blocks that present the sensors and the electronics that form the backbone of the test-rig. Hence, some parts may seem useless as we're providing the input then displaying the output after only rescaling it like the temperature but the data is used later in the Simulink model which will be seen in the following chapters.

4 Introduction to the algorithm

"In order to determine the characteristic pressure pulsation from the suction side of a hydraulic positive displacement pump, it is necessary to measure the pressure pulsation without a reflection of this pressure pulsation through the test stand (test rig).

The device for a measurement according to the ' RaLa ' procedure is represented in **Figure 23**. The inlet-side connection of the pump, that is to be measured, is connected via a straight pipe with the "RaLa". Along the pipeline, the sensors "p_pump stat", "p1_HDP dyn" and "p2_RaLa dyn" are installed, with which the mean pressure and the pressure pulsations in the pipeline can be measured.

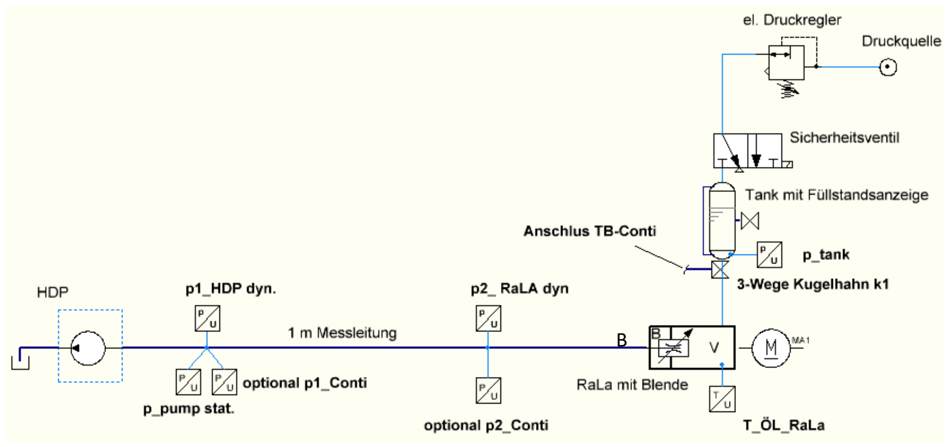


Figure 23: Schematic of the device to determine of the pumps pulsations.

The pipeline consists of the connector between the pump flange and the sensor "p1_HDP_dyn", the "measuring line" between the sensors of "p1_HDP_dyn" and "p2_RaLa_dyn" and the connector between the sensor "p2_RaLa_dyn" and the "RaLa". The "RaLa" consists of an aperture "B" and volume "V". The fluid temperature is included in the volume "V" using the temperature sensor "T_Öl_RaLa". By adjusting the aperture "B", you can set an (approximate) reflection-free state in the measuring line. The volume serves to prevent the reflections occurring within the "RaLa" from reaching the measuring line. At the outlet of the volume a gas storage tank (tank) is connected, over which with the help of an external pneumatic pressure supply, the desired pressure level in the measuring line can be adjusted. The measuring line and the two connecting pieces are made of a metal tube." (5)

4.1 Project Scope

The goal is to obtain fully automated positioning of the RaLa needle and an automatic start of the measurement.

During the project, this algorithm was improved by means of a combination/connection between test rig PLC and DSHplus simulation.

4.2 General Concept

After exporting the RaLa DSHplus module into a Simulink block, that includes the required sensors inside, we're going to exploit its outputs & take them as inputs to our Simulink scheme (**Figure 24**: *General Simulink model of the controller algorithm*).

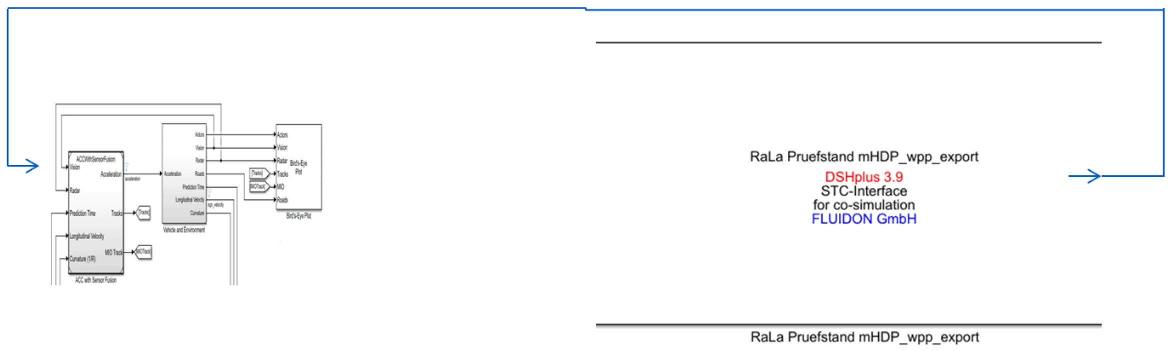

RaLa Pruefstand mHDP_wpp_export
DSHplus 3.9
STC-Interface
for co-simulation
FLUIDON GmbH
RaLa Pruefstand mHDP_wpp_export

Figure 24: General Simulink model of the controller algorithm

5 General Form

The following figure represents the general form of the inputs & outputs of the RaLa export module (**Figure 25**). It is important to point out that this was a preliminary structure of the model which was constructed as a first step of the whole approach.

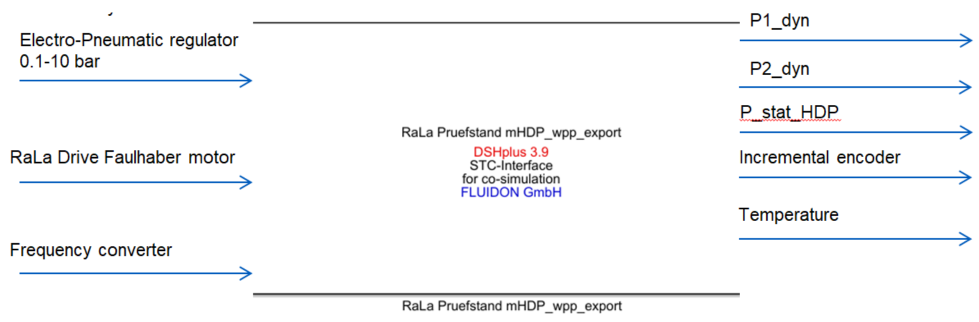


Figure 25: Inputs & outputs of the RaLa export module

5.1 Main Inputs

1) Pneumatic regulator: The electro-pneumatic regulator is represented as an input, so that when the user inserts the target system pressure, the regulator becomes active. In order to know if the target pressure is achieved, the signal coming from the P_sta_HDP must be controlled & thus input data must be inserted into the RaLa module & the results must pass through the output to insure that they are correct.
2) RaLa Faulhaber drive motor: It is the motor responsible of the needle's motion to reach the reflection-free state.
3) Frequency converter for the FCPA asynchronous motor which corresponds to the rotational speed (n).

5.2 Main Outputs

1) *P1_dyn & P2_dyn: 2 dynamic pressure sensors (transducers)*
2) P_stat_HDP: static pressure transducer
3) Incremental encoder signal: 512 pulses per revolution.
4) Temperature sensor: outputs the temperature.

6 Controller algorithm

Figure 26 represents the first brick of the Simulink model that is used to control the needle of the RaLa. The following figure is the exported DSHplus model with the corresponding inputs & outputs.

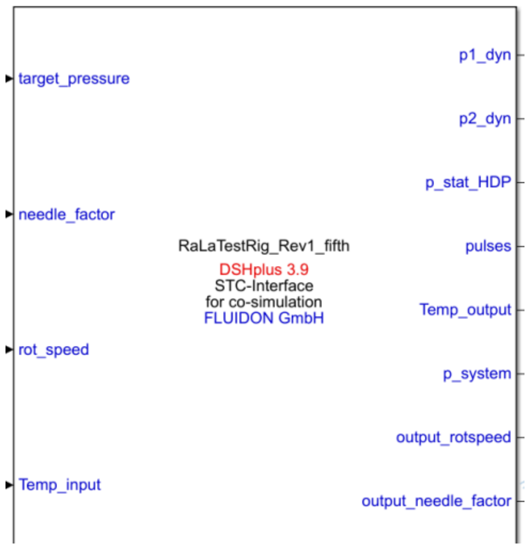


Figure 26: Exported DSHplus Simulink model

6.1 Inputs

1) Target pressure: represents the system static pressure(relative) (bar)
2) Needle factor: It is a variable in this equation:

$$\frac{\sqrt{bulk_{modulus} * oil\_density}}{\left({(pipe\_diameter)^2}/{4\pi}\right) * \left({needle\_factor}/{2}\right)}$$

   Its value ranges from $0^+ \rightarrow 2$. “0” for “fully_closed RaLa” where it acts as a closed piped & “2” for “fully_open” RaLa.

3) Rot_speed: represents the pump’s rotational speed (rpm)
4) Temp_input: represents the input temperature of the oil. This input is only used for simulation as on the real test rig we don’t have this input because we only have the output temperature of the oil on the test rig coming from the temperature sensor. (⁰C)

6.2 Outputs

1) P1_dyn & P2_dyn: 2 dynamic pressure oscillations from the 2 dynamic pressure sensors.(bar)
2) P_stat_HDP: static pressure from the static pressure sensor.(bar)
3) Pulses: 512 pulses/revolution
4) Temp_output: it represents the output temperature of the oil on the test rig. (⁰C)
5) P_system: output system pressure to make sure that the system pressure value has hit the target pressure. (bar)
6) Output_rotspeed: output rotational speed to make sure that the rotational speed value has hit the assigned rotational speed of the pump. (rpm)
7) Output_needle_factor: the output value of the needle factor.

6.3 Low pass filter

It is used to pass signals with a frequency lower than the selected cutoff frequency which is 800 Hz here and attenuates signals with frequencies higher than 800 Hz. A first order low pass filter is chosen for both signals coming from p1 & p2 dynamic pressure transducers. (**Figure 27**)

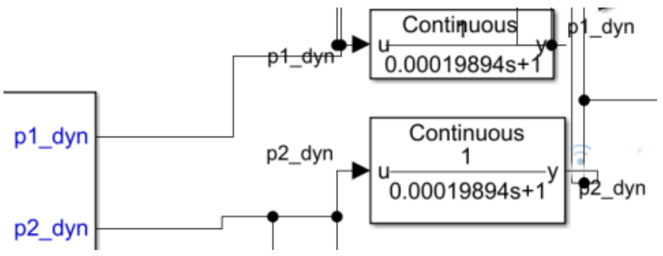
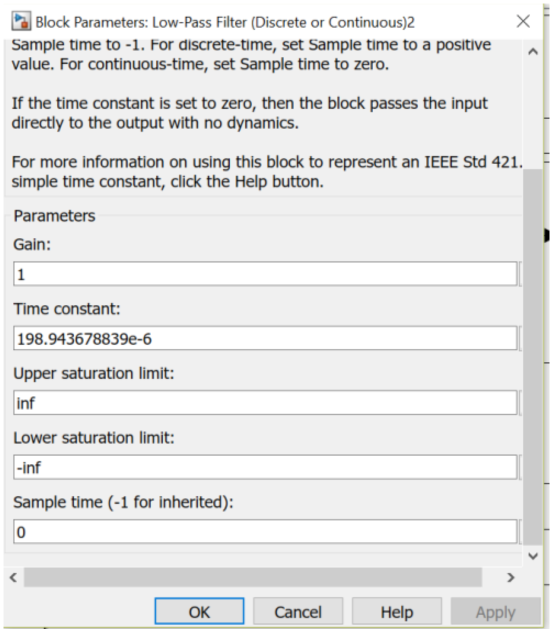


Figure 27: Low pass filters of p1_dyn & p2_dyn pressure signals

The block paramters to be filled when choosing the “Low-Pass Filter (Discrete or Continuos)” block in Simulink can be seen in **Figure 28 .**



Block Parameters: Low-Pass Filter (Discrete or Continuous)2

Sample time to -1. For discrete-time, set Sample time to a positive value. For continuous-time, set Sample time to zero.

If the time constant is set to zero, then the block passes the input directly to the output with no dynamics.

For more information on using this block to represent an IEEE Std 421. simple time constant, click the Help button.

Parameters

Gain: 1

Time constant: 198.943678839e-6

Upper saturation limit: inf

Lower saturation limit: -inf

Sample time (-1 for inherited): 0

Figure 28: Low-Pass filter parameters

Notes:

- Time constant=τ=$\frac{1}{2\pi f}$
- Sample time=0 for continuous signals

6.4 Time delay

Due to the 1 m distance between the two pressure transducers, **p1_dyn** & **p2_dyn**, a time offset (delay) can be seen between the corresponding pressure pulsations. On the test rig, this time delay doesn’t have any significant effect on the results, but this offset has to be compensated on **p1_dyn** signal in order to be able to compare the signals in an accurate way & for correct implementation of the controller algorithm. **Figure 29**

This delay is implemented in Simulink by means of a ″transport delay” block with time delay=$\frac{1}{\sqrt{\frac{bulk\ modulus}{density}}}$

Bulk modulus=11526.1 bar

Density=756.4 kg/m^3

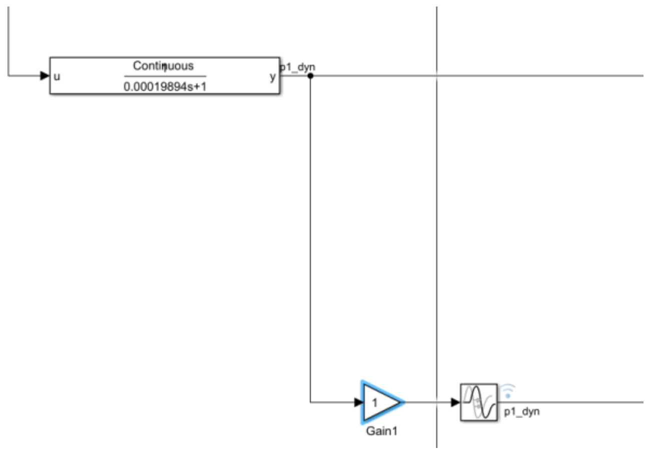


Figure 29: Time offset imposed on p1_dyn signal

PS: A gain equal to "1" had to be added before the delay block because the output signal always resulted zero before inserting it.

6.5 The basic control approach

After making sure that a suitable time delay is imposed on the ***p1_dyn*** signal in order to avoid any lag, it is time to start thinking of a robust & feasible control approach.

Objective:

Matching ***p1_dyn*** & ***p2_dyn*** signals with the peak of ***p1_dyn*** slightly higher than that of ***p2_dyn*** because position of the latter comes 1m after the ***p1_dyn*** transducer, thus some damping will take place that will result in a slight decrease of ***p2_dyn's*** maximum.

Approach:

The main approach is divided into three main steps:

i. Finding the maxima of ***p1_dyn*** & ***p2_dyn*** signals (bar or Pa) & subtracting the latter from ***p1_dyn***. Our goal is to minimize the difference to stay within a pre-specified limit.
ii. Finding the intersection of both signals with the x-axis (seconds) & subtracting that of p2_dyn from p1_dyn. Our goal is to minimize the difference to stay within a pre-specified limit.
iii. When the previous conditions are satisfied for a specific number of times, the needle factor is to be fixed & the optimum reflection-less condition is reached.

6.6 Detecting the maxima

The algorithm to find the maxima is composed of 2 subsystems & a trigger block that are joined together in a parent subsystem called “p1_max” (***p1_dyn***) & “p2_max” (***p2_dyn***). **Figure 30** & **Figure 31**

PS: The inputs of p1_max & p2_max are ***p1_dyn*** & ***p2_dyn*** signals respectively, while the outputs are the values of the maxima.

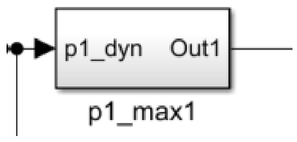


Figure 30: parent subsystem

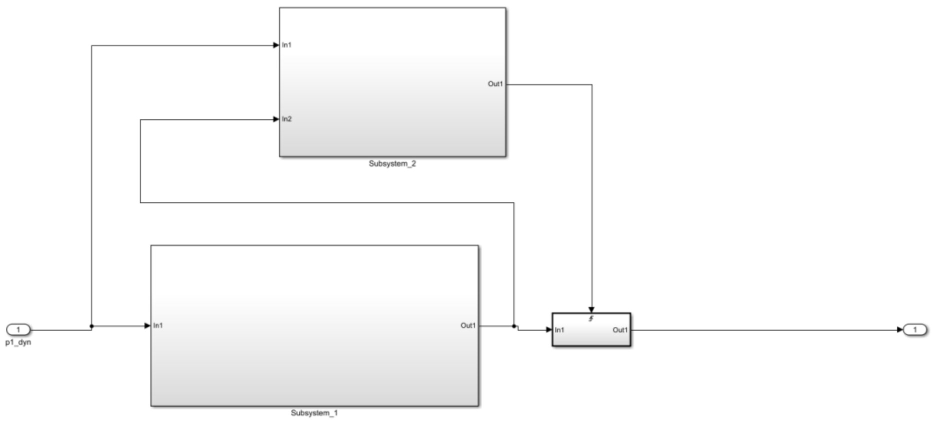


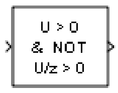
Figure 31: Expanded form

6.6.1 Subsystem_1

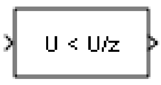
It is composed of the following blocks:

i. MinMax Running Resettable: Outputs the max or min (chosen function is max) of all past inputs u. The output is reset to the initial condition when the Reset input signal R is TRUE. This reset action is vectored and supports scalar expansion.

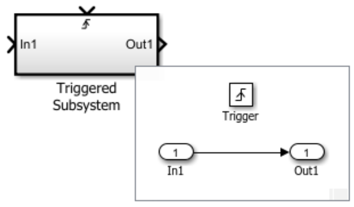
ii. Detect Rise Positive: If the input is strictly positive and its previous value was non-positive, then output TRUE, otherwise output FALSE. The initial condition determines the initial value of the Boolean expression (U/z > 0).



iii. Detect decrease: If the input is strictly less than its previous value, then output TRUE, otherwise output FALSE. The initial condition determines the initial value of the previous input U/z.



iv. The triggered subsystem: The Triggered Subsystem block is a Subsystem block preconfigured as a starting point for creating a subsystem that executes each time the control signal has a trigger event.



What basically happens in this subsystem is that the “minmax resettable” block outputs the max of all past inputs u & resets whenever it receives a Boolean value of “1” (TRUE) from the “detect rise positive block” (i.e. whenever the pressure signal crosses the x-axis.

The maximum values are fed as inputs to “the triggered subsystem” that outputs a specific value whenever it is triggered, or in other words, receives a Boolean value of “1” (TRUE) from the “detect decrease” block. **Figure 32**

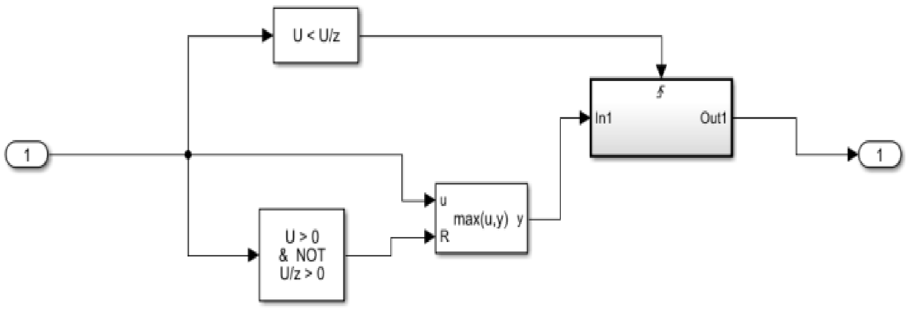


Figure 32:Subsystem_1

6.6.2 Subsystem_2

If we take the previous subsystem only to give us the maxima, the results will be as in **Figure 33** which is unacceptable as the values of the maximum aren’t held waiting for a new value, but they become zero whenever the “***minmax***” block resets. In addition to that, all the values of the p1_dyn & p2_dyn rising edges from the x-axis to the peak are considered as maxima which is wrong, thus we need subsystem_2.

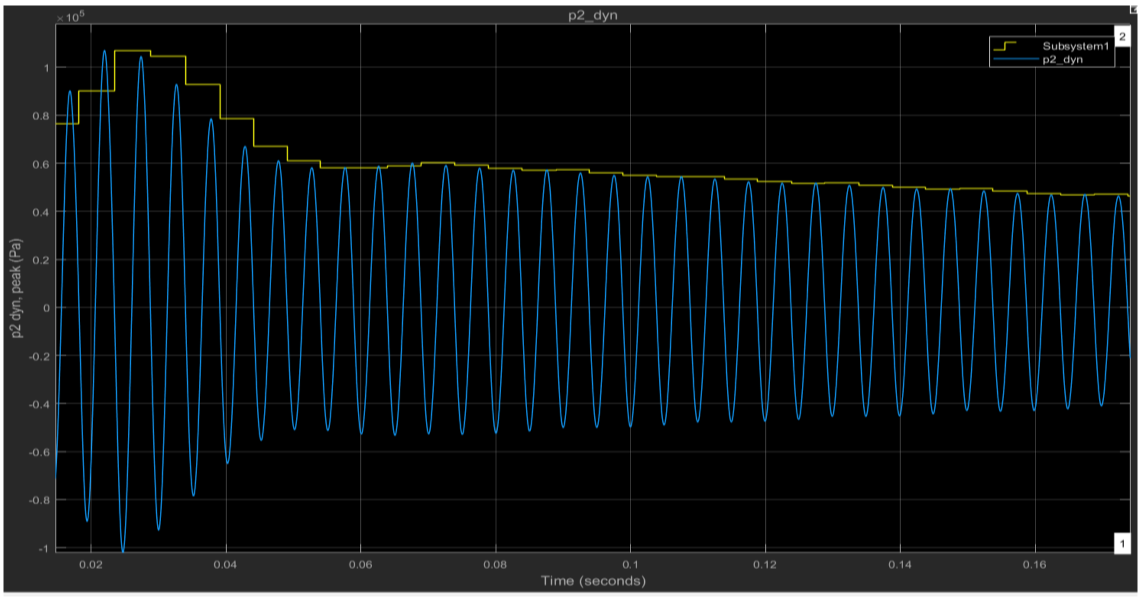
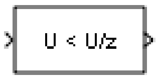


Figure 33: *Detecting the maxima*

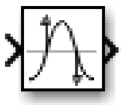
Subsystem_2 is composed of the following blocks:

i. MinMax Running Resettable: Outputs the max or min (chosen function is max) of all past inputs u. The output is reset to the initial condition when the Reset input signal R is TRUE. This reset action is vectorized and supports scalar expansion.

ii. Detect decrease: If the input is strictly less than its previous value, then output TRUE, otherwise output FALSE. The initial condition determines the initial value of the previous input U/z.



iii. Hit Crossing: Detects when the input signal reaches the Hit crossing offset parameter value in the direction specified by the Hit crossing direction parameter. If the input signal crosses the offset value in the specified direction, the block outputs 1 at the crossing time. If the input signal reaches the offset value in the specified direction and then remains at the offset value, the block outputs 1 from the hit time till the time when signal leaves the offset value. If the input signal is constant and equal to the offset value, the block outputs 1 only if the direction is either. For variable-step solvers, Simulink takes a time step before and after the hit crossing time.



The inputs are:

i. In1: ***P1_dyn*** or ***p2_dyn*** (depending on which maxima are we trying to obtain).

ii. In2: The output values from subsystem_1.

So what happens is that the output values of subsystem_1 are fed as inputs to the "minmax resettable" block with "max function" & the "hit crossing" block resets the latter every time the pressure signal hits the x-axis. Then these values are fed to a "detect decrease" block that outputs "1" every time a decrease in these values is detected. **Figure 34**

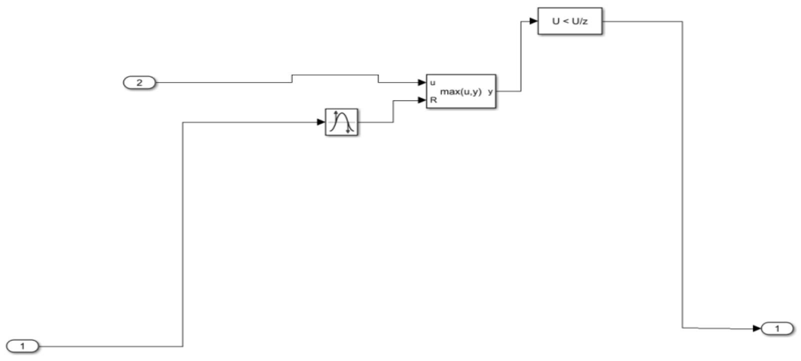


Figure 34: Subsystem_2

The output of the "detect decrease" block is connected to the trigger of a "triggered subsystem" block outside subsystem_2. This block that can be seen in **Figure 31** is fed, as input, the output values of subsystem_1 while subsystem_2 is used to trigger it.

6.6.3 Additional Subsystem

An extra subsystem is added after the parent subsystem p1_max/p2_max. This subsystem rejects all maxima with values less than a specific threshold. The reason is that sometimes due to some noise or due to the characteristic behaviour of the pump, a local maximum with a small value can satisfy all the conditions of the algorithm, in rare cases, which can cause an unwanted delay & mess in the controller algorithm.

The "threshold_max" subsystem is composed of the following blocks:

i. If Block: equivalent to the "if statement" in the other programming languages.
ii. If Action Subsystem: The If Action Subsystem block is a Subsystem block preconfigured as a starting point for creating a subsystem whose execution is triggered by an If block.

As seen in **Figure 35,** a specific value is considered as a maximum only if it's greater than 0.15 bar.

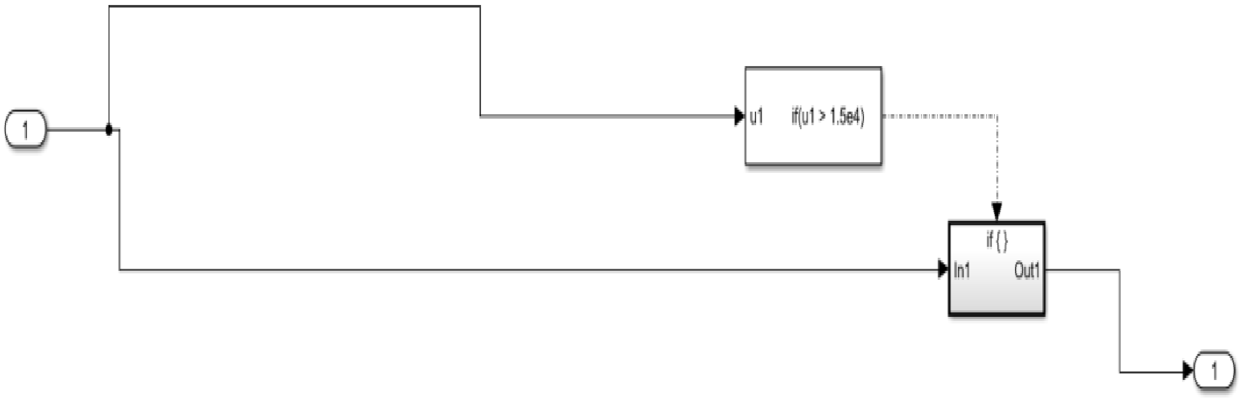


Figure 35:Threshold_max

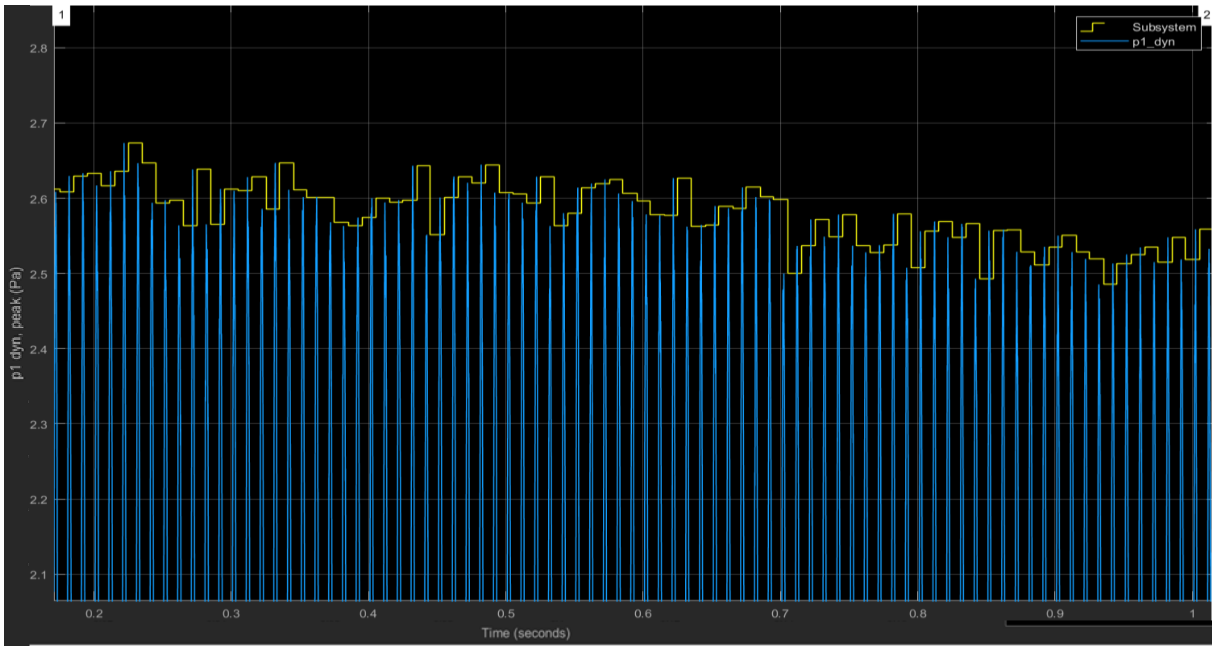


Figure 36: peak detection for the p1_dyn signal

Results: The final results for detecting the maxima of ***p1_dyn*** signal can be seen in **Figure 36**

6.7 Finding the x-intercept

Figure 37 shows both subsystems that give the x-intercepts of ***p1_dyn*** & ***p2_dyn*** respectively & sub-tracts them.

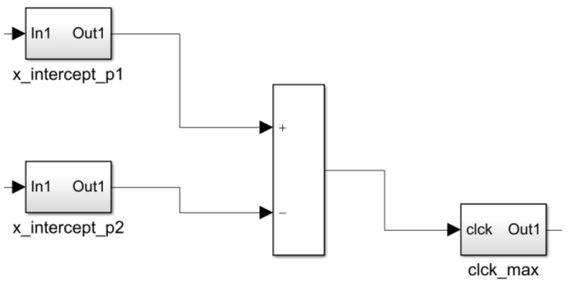
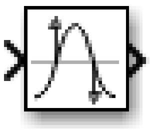


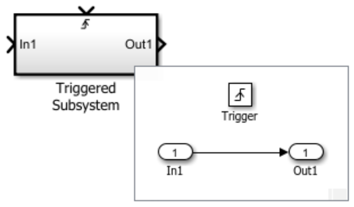
Figure 37: difference between x-intercepts

The two x-intercept subsystems are equivalent; every subsystem is composed of the following blocks:

i. Hit Crossing: Detects when the input signal reaches the Hit crossing offset parameter value in the direction specified by the Hit crossing direction parameter. If the input signal crosses the offset value in the specified direction, the block outputs 1 at the crossing time. If the input signal reaches the offset value in the specified direction and then remains at the offset value, the block outputs 1 from the hit time till the time when signal leaves the offset value. If the input signal is constant and equal to the offset value, the block outputs 1 only if the direction is either.



ii. The triggered subsystem: The Triggered Subsystem block is a Subsystem block preconfigured as a starting point for creating a subsystem that executes each time the control signal has a trigger event.



iii. Clock: Output the current simulation time.

As it can be seen in **Figure 38** the "triggered subsystem" is fed the simulation time as an input but it only outputs the corresponding simulation time when it hits the x-axis (zero crossing) either in the rising edge or the falling one.

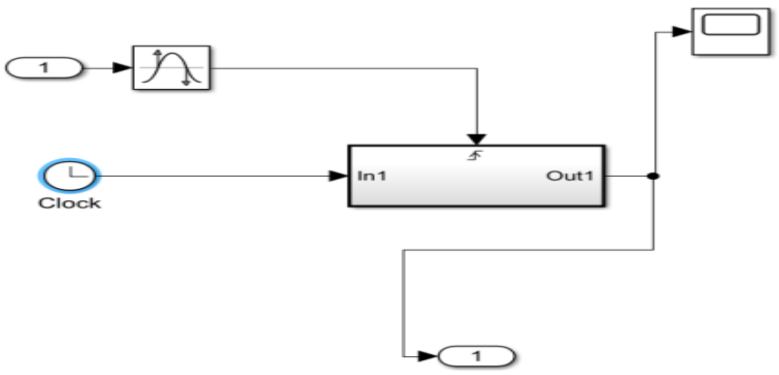


Figure 38: x-intercept of p1_dyn signal

The "clck_max" block that can be seen after obtaining the difference between the x-intercepts in **Figure 36** is composed of the same blocks as the p1_max & p2_max blocks in **Figure 38.**

This block is used to differentiate the x-intercept of every oscillation in the pressure signal from the noise or fluctuations due to the characteristic behavior of the pump.

6.8 MATLAB Function & stateflow

6.8.1 The MATLAB Function

The general idea is that when a set of conditions are satisfied, a variable "y" which is given an initial value of "0" becomes equal to "1". This process takes place through a simple "if statement". Along with the MATLAB function there is a counter which counts the number of times the variable "y" becomes "1".

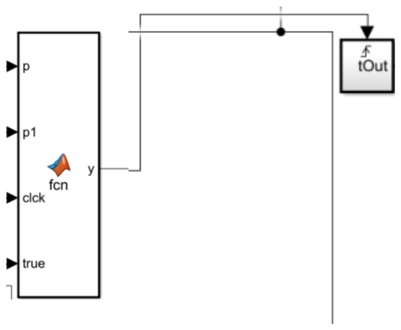


Figure 39: The MATLAB function & the counter

As seen in **Figure 39** the inputs of the MATLAB function are:

i. p: which is the difference between the maxima of ***p1_dyn*** & ***p2_dyn*** signals (i.e.: p1_max-p2_max for every oscillation)
ii. p1: represents ***p1_dyn*** signal
iii. clck: which represents the difference between the x-intercepts of the 2 dynamic pressure signals ***p1_dyn*** & ***p2_dyn*** (i.e.: x-intercept (***p1_dyn***) - x-intercept (***p2_dyn***))
iv. True: if both the rotational speed & target pressure conditions were satisfied (i.e. when both of them are within range. **Example:** *output_pressure= target_pressure* ± tolerance.)

- The only output of the MATLAB function is the variable y which is fed as an input to the counter.

The conditions are:

i. 100 Pa $\leq \boldsymbol{p} \leq$ 600 Pa
ii. ***clck*** $\leq$ 0.008 seconds
iii. ***p1*** $\geq$ 0.1 bar

***Figure* 40** represents the counter used to count the number of times “y” becomes equal to “1”.

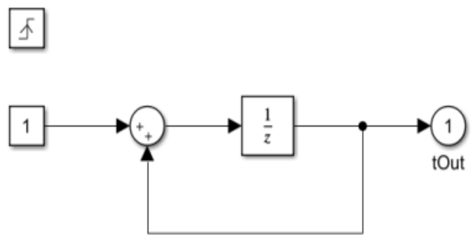


Figure 40: The counter of “y”

6.8.2 Stateflow Chart

This part is the final & most important part of the controller algorithm. What contributes in its importance is the fact that it is the part responsible of finding the optimum needle factor that will satisfy the previous conditions for a certain number of times.

The basic idea is to increase the needle factor by small increments whenever p<150 Pa & decrease it by small decrements whenever p>600 Pa. In case the previous conditions are satisfied for a minimum of 5 times (for 5 oscillations not necessarily consecutive), then the needle factor is fixed on the last value. **Figure 41**

Inputs:

- The output coming from the counter block that counts the number of times “y” becomes TRUE.
- p: the difference between the maxima of ***p1_dyn*** & ***p2_dyn*** signals.
- True: It is explained in **Figure 42**.
- Mode: which is basically a Boolean that is True (1) in case automatic mode is chosen, & False (0) in case manual mode is chosen.

Local data:

- The variable “n1” is of type: “local data” that is only used to initialize the value of the needle factor “n” & updating its value through the iterations.

Outputs:

- The only output is the needle factor “n” that is passed to the DSHplus exported model which will, by its turn, change the position of the needle adjusting the pressure pulsations to finally match each other.

- Fixed: It only serves as a flag to signalize that the needle factor is fixed & the algorithm was successful.

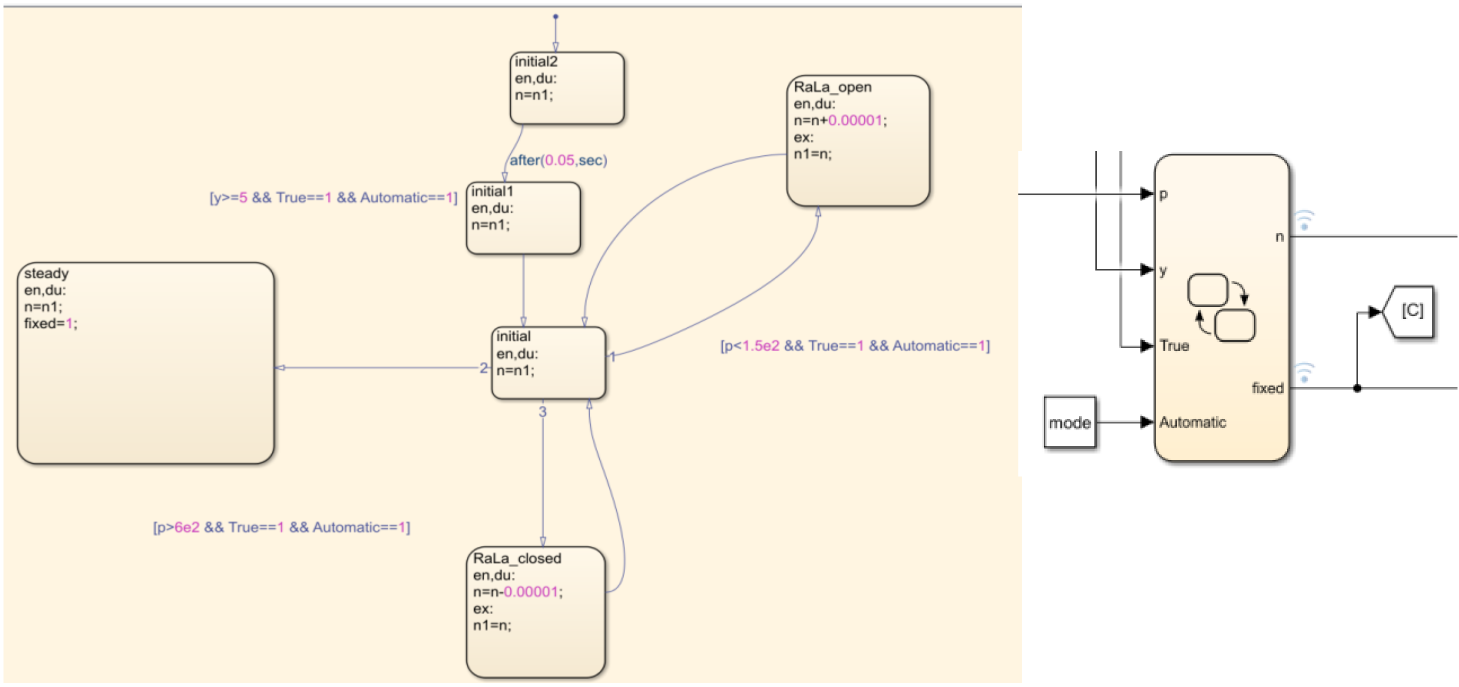


Figure 41: Stateflow Chart that contains a significant part of the controller algorithm

- The first two "initial" blocks are used to delay the start of the calculations in order to give some time, even though it might be relatively small, for the system to stabilize & deliver the expected results.
- The third "initial" block is the center block or the "decision maker", in which, the controller waits to process the data & move to the suitable block (either increase or decrease the needle factor).
- Eventually, when the conditions are satisfied for a specific number of times (five in our case), the needle factor is fixed & the next step is to take place (depending on the user's choice).

7 HMI (Human Machine Interface)

Objective: Designing a part of computer program that communicates with the user in order for this user interface to act as an interactive system (software or hardware) that provides information and control what's necessary for the user to complete a certain task with this system.

How does the HMI look like?

The approach that is to be followed to create the HMI is contingent to our visualizations of how the user is going to exploit this interface, so basically it depends on the developer's insight.

In simple words, the next few lines describe the steps that the user has to follow in order to start running the test rig:

1) The user is asked to insert the system target pressure (static pressure) where the corresponding pneumatic regulator is going to insure this target pressure (bar) is reached.
2) The user is then asked to insert the desired rotational speed (rpm).
3) It is important to mention that neither the controller algorithm in automatic mode nor the needle adjustment in the manual mode is going to work if the *output_pressure* or the rot_speed aren't within a specific range.
4) Insert the low pass filter frequency (800 Hz is recommended)
5) Choose between automatic or manual mode for the adjustment of the needle. In the automatic mode the needle is shifted automatically to a position determined according to the controller algorithm. In the manual mode the needle is adjusted by the user.
6) The user is required to choose between two operation modes:
   a. **DAQ Autostart**: Start saving the *p1_dyn* & *p2_dyn* measurement data automatically after the optimum position of the needle is reached, according to the controller algorithm, & the saving process is halted when the user stops it.
   b. **DAQ Autostop**: Start saving the *p1_dyn* & *p2_dyn* measurement data automatically after the optimum position of the needle is reached, according to the controller algorithm, & the saving process is halted when the pre-specified number of revolutions is reached.

Understanding a specific algorithm or piece of code can be quite troublesome when seeing a big number of blocks enclosed in a small subsystem that seems simple from the first look. For this reason, I'm going to walk you through every single step in order to guarantee a better understanding of every command & how it is executed in the background.

7.1 Target Pressure & rotational speed

As mentioned before, neither the controller algorithm in automatic mode nor the needle adjustment in the manual mode is going to work if the *output_pressure* or the *rot_speed* aren't within a specific range. What is meant by a specific range is a tolerance that can be inserted as a variable or a constant.

- *Output_pressure*: For the target pressure I imposed a condition on the system that the *output_pressure* has to be within a ± 0.1 bar range.
- *Rot_speed* For the rotational speed I imposed a condition on the system that the *output_rotspeed* has to be within a ± 10 rpm range.

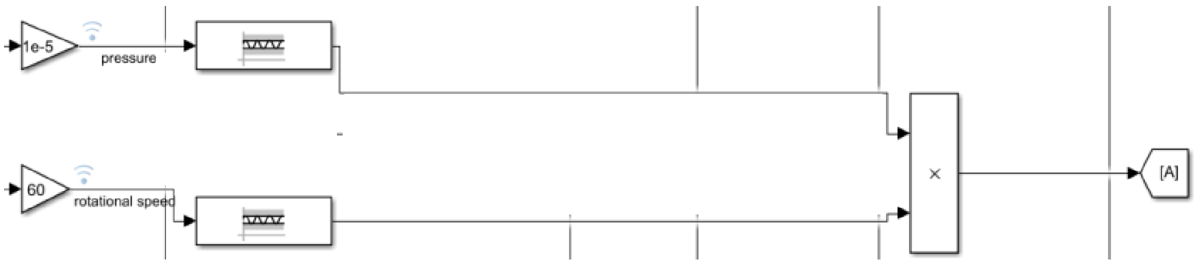


Figure 42: Pressure & rotational speed range verification

Figure 42 represents the system of blocks used to verify that both the output pressure & rotational speed are within the specified range. In order to do that the following blocks are used:

- The "***gain***" block which is multiplied by both signals to convert the pressure values from Pa to bar & the rotational speed from rps to rpm.
- ***Check static range***: Asserts that the input signal lies between a static lower and upper bound or optionally equals either bound by producing a Boolean number that is either 1 (True) or 0 (False).

- ***Product***: To multiply inputs, where the outputs of both static range blocks are multiplied & the result is 0 (False) if one of the static range blocks outputs is false & 1 (true) if & only if both are true.
- ***Goto***: Send signals to "From" blocks that have the specified tag. This block passes the output of the *product* block to the "From" block which passes this value to both the MATLAB function (**Figure 39**) & stateflow chart (**Figure 41**) through the variable "*True*".

7.2 Automatic Mode

In this mode, the needle is adjusted & shifted to the right position automatically through the algorithm that we went through before.

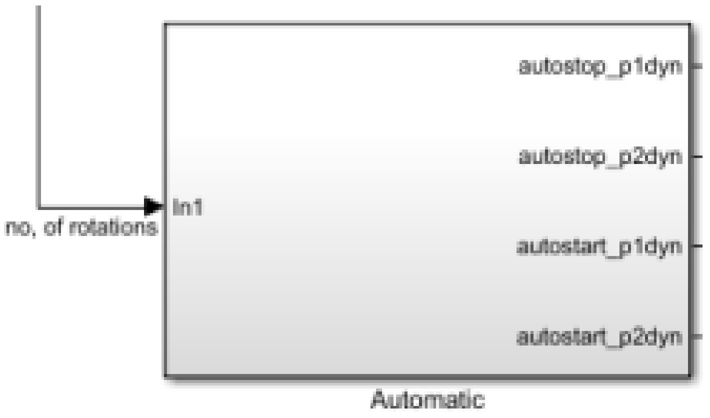


Figure 43: The subsystem that contains the automatic mode algorithm

The subsystem in **Figure 43** contains the algorithm that manages the transfer of data, depending on the chosen operation mode**,** to the assigned server or scope & takes care of executing the proper actions.

- The input seen in **Figure 43** is the number of rotations that is calculated starting from the moment in which the needle factor becomes fixed.

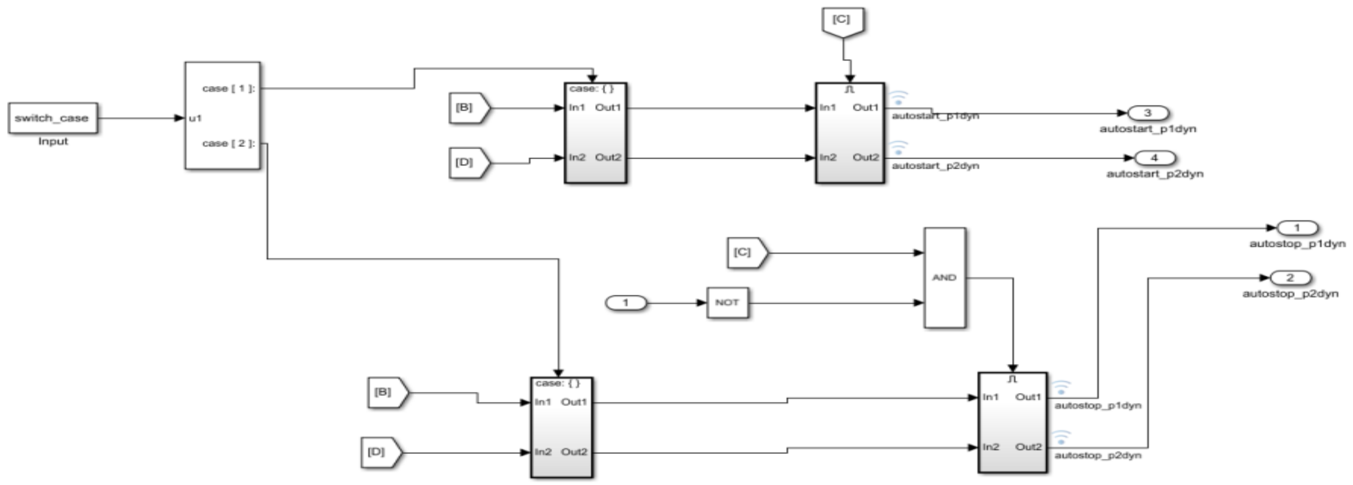


Figure 44: An inside look at the automatic mode subsystem

The switch case in **Figure 44** determines, according to user's choice, the series of commands that are going to be executed, that can either be:

- Case (1): DAQ Autostart
- Case (2): DAQ Autostop

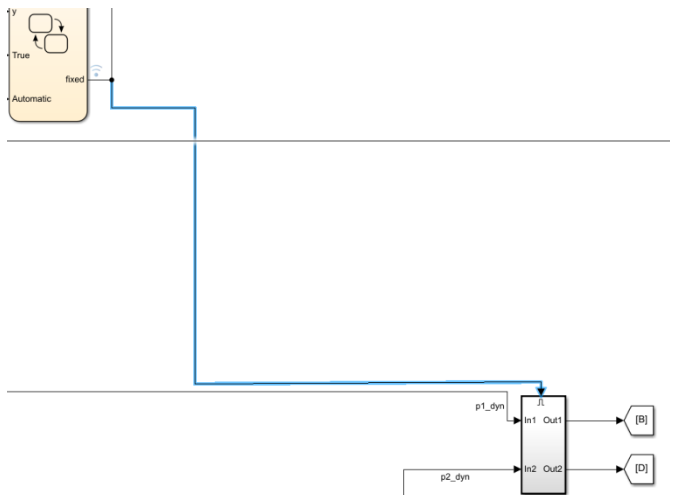


Figure 45: signals connected to **B** & **D** *Goto* blocks

It is important to emphasize on the role of the two signals that can be seen in **Figure 45**, their importance lies in the fact that these two signals are the ones to be exploited by the two DAQ operation modes when the needle factor becomes fixed (i.e. when the control algorithm finds the right position of the needle).

How does this process work?

The subsystem seen in **Figure 46** is an enabled subsystem where it is a subsystem whose execution is enabled by external input. It is used to enable the passage of the *p1_dyn* & *p2_dyn* blocks whenever the variable ***fixed*** is true (its functionality was discussed under the title **outputs** in the **6.8.2 Stateflow Chart section**).

7.2.1 DAQ Auto start

Summary: When the controller algorithm finds the optimum position of the needle & the needle factor becomes fixed, a flag signalizes this event which by its turn notifies the responsible subsystem which outputs the *p1_dyn* & *p2_dyn* signals until the user stops this action.

Reminder: As mentioned before, when choosing this operation mode, the test rig will start saving the *p1_dyn* & *p2_dyn* measurement data automatically after the optimum position of the needle is reached, according to the controller algorithm, & the saving process is halted when the user stops it.

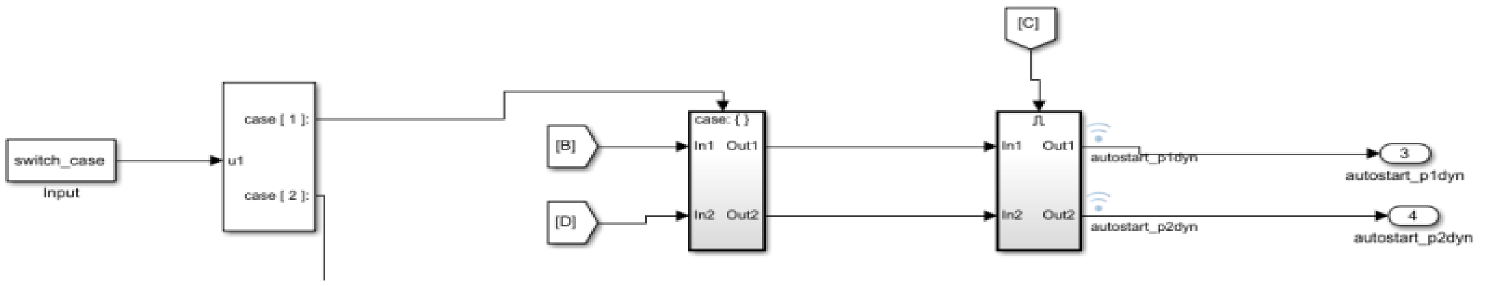
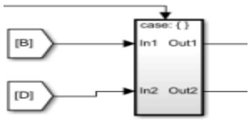


Figure 46: DAQ Auto start (Automatic)

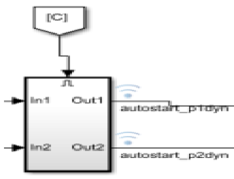
How does it work?

1) When the user chooses this operation mode, the variable *switch_case*'s value changes to **1.**

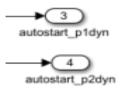
2) The corresponding switch case action subsystem becomes active, it's a subsystem whose execution is triggered by switch block & it has two inputs: ***B*** & ***D*** signals of block type "**From**".



3) The output of this subsytem forms the input of the enabled subsystem. This susbsystem is enabled by the ***C From*** block which receives the signal from the corresponding ***C Goto*** block that is connected to the output ***fixed*** of the stateflow chart.



4) The output is then sent via these output blocks to the server that stores these measurement data or to a scope that displays them.



7.2.2 DAQ Auto stop

Summary: The basic idea is that when the controller algorithm finds the optimum position of the needle & the needle factor becomes fixed, a flag signalizes this event which by its turn notifies the responsible subsystem which outputs the *p1_dyn* & *p2_dyn* signals until the number of rotations reaches the value assigned by the user.

Reminder: When choosing this operation mode, the test rig will start saving the p1_dyn & p2_dyn measurement data automatically after the optimum position of the needle is reached, & the saving process is halted when the pre-specified number of revolutions is reached.

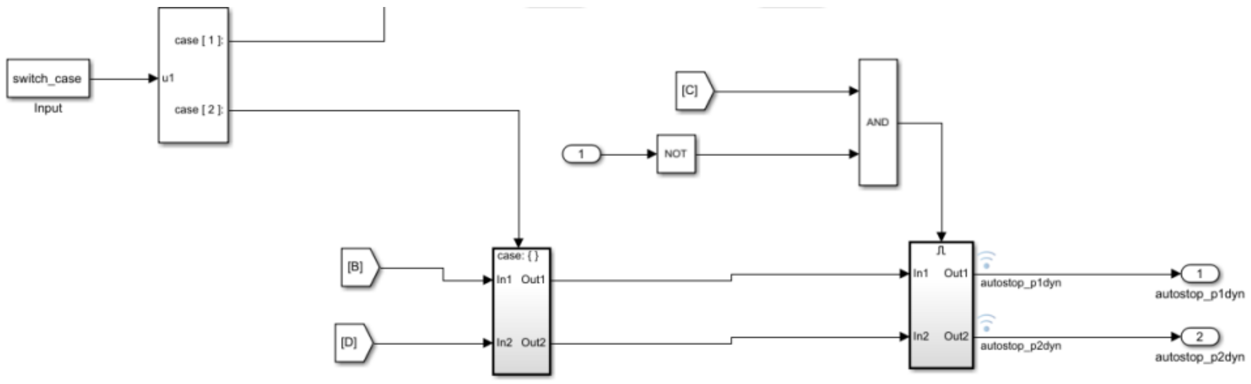
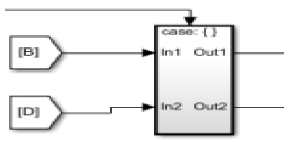


Figure 47: DAQ Auto-stop (Automatic)

How does it work?

1) When the user chooses this operation mode, the variable *switch_case*'s value changes to **2.**
2) The corresponding switch case action subsystem becomes active.



3) The output of this sybsytem forms the input of the enabled subsystem. This subsystem must be enabled when the needle factor becomes fixed, only then it can start counting the number of rotations from that point until it reaches the value assigned by the user. In order to achieve this, a block that executes the "AND" logic is one responsible of enabling or disabling the subsystem. The inputs of the "AND" block are the current value of the variable *fixed* & the negation of the no. of rotations verification blocks (**Figure 48**).

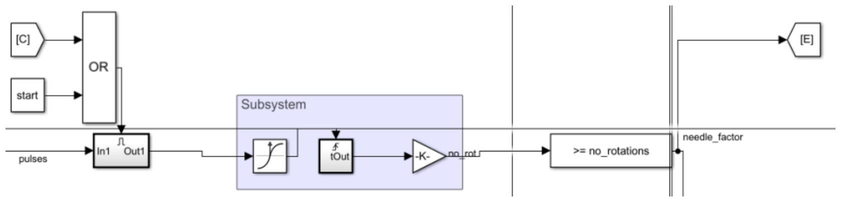
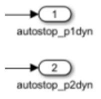


Figure 48: The number of rotations verification blocks

As seen in **Figure 48,** the input of the enabled subsystem is the pulses signal coming from the DSHplus model & the external input that triggers its execution is the output of the "OR" block (for the automatic mode the only input of this block that we care about is the input "***C From***" block that represents the value of the "***fixed***" variable). So when the system gets informed that the needle factor is fixed, it starts allowing the passage of the ***pulses*** signal that comes in the form of **1** or **0.** The ***hit crossing*** block detects every time the ***pulses*** signal is one & feeds it to the counter that starts counting the number of pulses. This number is divided by 512 because our incremental encoder produces 512 pulses per revolution. The number of revolutions is then fed to the "***compare to constant***" block that gives "TRUE" whenever the number of revolutions is greater than the value of the variable "***no_rotations***" inserted by the user.

4) The output is then sent via these output blocks 

to the server that stores these measurement data or to a scope that displays them. They are sent during the period when two conditions are satisfied together:
   - Needle factor is fixed.
   - The number of revolutions or rotations is less than or equal to the one inserted by the user.

7.3 Manual Mode

In this mode, the needle is adjusted & shifted to the desired position through a hand wheel where the user keeps track of the output signals through a scope displaying them on the screen. When the needle reaches the desired position, the user presses on the "**start**" button to start saving the measurement data & stops this process according to the chosen case by the user:

- Case (1)-DAQ Autostart: halts the saving process when the user stops it.
- Case (2)-DAQ Autostop: after a specific number of rotations determined by the user before starting the measurements.

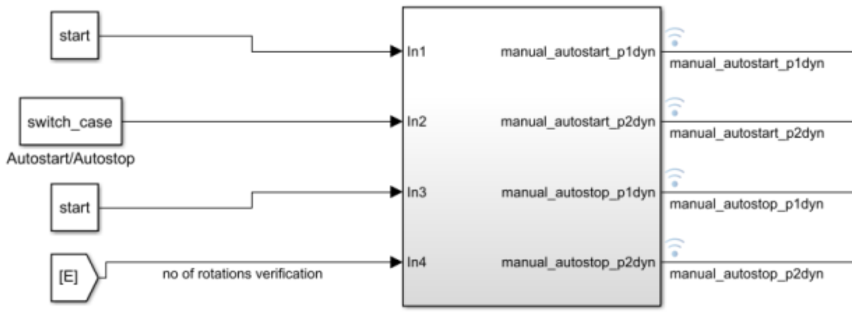


Figure 49: The subsystem that contains the manual mode algorithm

The subsystem in **Figure 49** contains the algorithm that manages the transfer of data, depending on the chosen operation mode**,** to the assigned server or scope & takes care of executing the proper actions.

Inputs:

- ***Start***: It can be either False (0) or True (1) and is activated when the user presses the start button on the HMI .
- ***Switch_case*****:** Holds the values 1 or 2 depending on the user's choice of the operation mode (DAQ **Autostart** or **Autostop).**
- ***(E)***: "From" block that is "TRUE" when the number of rotations, after the needle factor becomes fixed, is greater than the number of rotations assigned by the user on the interface.

Outputs:

- ***P1_dyn*** & ***P2_dyn*** signals in the DAQ Autostart mode.
- ***P1_dyn*** & ***P2_dyn*** signals in the DAQ Autostop mode.

7.3.1 DAQ Auto start

Summary: When the user finds the desired position of the needle & presses the start button on the HMI, this activates the responsible subsystem which outputs the *p1_dyn* & *p2_dyn* signals until the user stops this action.

Reminder:

As mentioned before, when choosing this operation mode, the test rig will start saving the *p1_dyn* & *p2_dyn* measurement data after the user presses the "**start**" button & the saving process is halted when the user stops it.

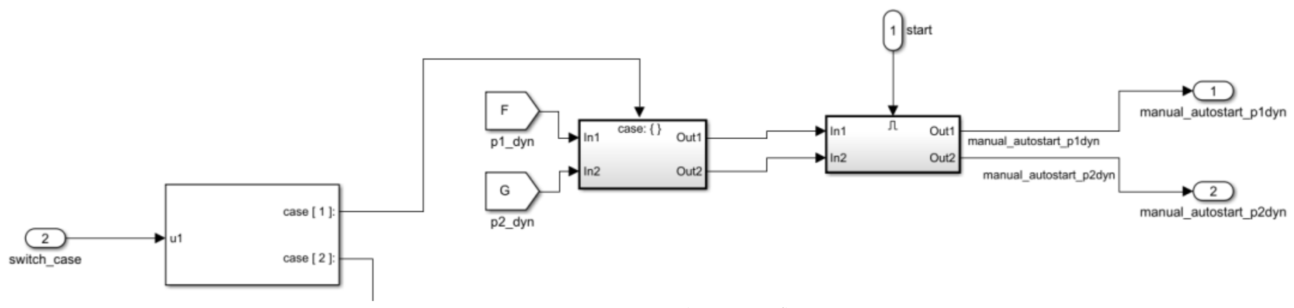
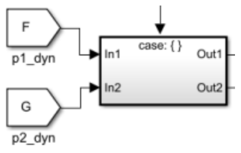


Figure 50: DAQ Auto start (Manual)

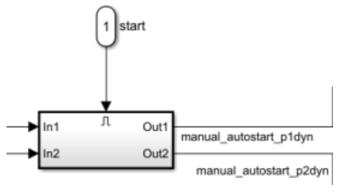
How does it work?

As seen in **Figure 50**:

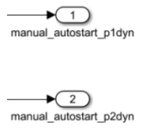
1) When the user chooses this operation mode, the variable *switch_case*'s value changes to **1.**



2) The corresponding switch case action subsystem becomes active, it's a subsystem whose execution is triggered by switch block & it has two inputs: ***F*** & ***G*** signals of block type "**From**".
3) The output of the switch case subsystem is the input of the following enabled subsystem. This subsystem is enabled by the ***start*** block which is the variable that responds to the start button on the user's interface.



4) The output is then sent via these output blocks to the server that stores these measurement data or to a scope that displays them.



7.3.2 DAQ Auto stop

Summary: When the user finds the desired position of the needle & presses the start button on the HMI, this activates the responsible subsystem which outputs the *p1_dyn* & *p2_dyn* signals until the number of rotations reaches the value assigned by the user.

Reminder: As mentioned before, when choosing this operation mode, the test rig will start saving the *p1_dyn* & *p2_dyn* measurement data after the user presses the "**start**" button & the saving process is halted when the number of rotations reaches the one assigned by the user.

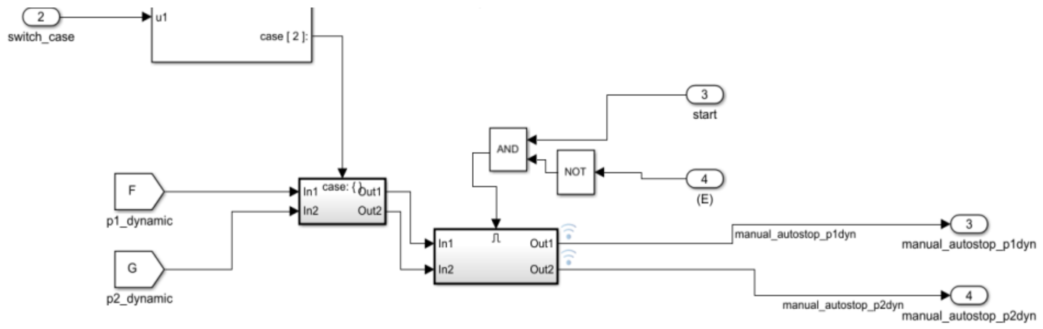


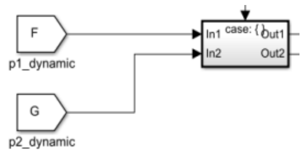
Figure 51: DAQ Autostop (Manual)

How does it work?

As seen in **Figure 51**:

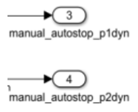
1) When the user chooses this operation mode, the variable *switch_case*'s value changes to **2.**

2) The corresponding switch case action subsystem becomes active.



3) The output of this subsystem forms the input of the enabled subsystem. This subsystem must be enabled when the user presses the **start** button, only then it can start counting the number of rotations from that point until it reaches the value assigned by the user. In order to achieve this, a block that executes the "AND" logic is one responsible of enabling or disabling the subsystem. The inputs of the "AND" block are the current value of the variable *start* & the negation of the no. of rotations verification blocks (**Figure 51)**.

As seen in **Figure 51,** the input of the enabled subsystem is the pulses signal coming from the DSHplus model & the external input that triggers its execution is the output of the "OR" block (for the manual mode the only input of this block that we care about is the "***start***" block). So when the system gets informed that the user activated the start saving button, it starts allowing the passage of the ***pulses*** signal that comes in the form of **1** or **0.** The ***hit crossing*** block detects every time the ***pulses*** signal is one & feeds it to the counter that starts counting the number of pulses. This number is divided by 512 because our incremental encoder produces 512 pulses per revolution. The number of revolutions is then fed to the "***compare to constant***" block that gives "TRUE" whenever the number of revolutions is greater than the value of the variable "***no_rotations***" inserted by the user.



4) The output is then sent via these output blocks to the server that stores these measurement data or to a scope that displays them. They are sent during the period when two conditions are satisfied together:
   - The start button is pressed or activated.
   - The number of revolutions or rotations is less than or equal to the one inserted by the user.

7.4 GUI (Graphical User Interface)

Definition: It is the interface which replaces cryptic commands by their graphical representation, it is a form of user interface that allows users to interact with electronic devices (simulation model in our case) through graphical icons and visual indicators such as secondary notation, instead of text-based user interfaces, typed command labels or text navigation.

Goal: The blocks in the Dashboard library in Simulink will help us control and visualize our Simulink model during simulation and while the simulation is paused in order to avoid the complications of inserting the values of the variables & observing the measurement data (output) from the numerous & complex blocks in our model.

How is it created?

1) Initialize all variables in the MATLAB editor by zero.
2) Create a subsystem that contains all the graphical representations & name it "graphical control displays".
3) The displays (switches, gauges, buttons, scopes, knobs, etc..) are to be divided into two groups:
   a. Parameters: It contains the values to be inserted for the input variables, operation modes (Automatic/Manual, Auto start/Auto stop) & the start button that is used in the manual mode. **Figure 52**

Outputs: It contains the gauges & the scopes to display the measurement data in response to our inserted data. Connect the ″Dashboard blocks″ to their respective variables & signals. **Figure 53**

How does it work?

Simply, click on the block that you need to modify & insert the desired value, move the slider, turn on the switch or rotate the knob to the desired mode. After carrying out all the required actions correctly, the results should appear on the gauges, scopes & displays shown in ***Figure 53***.

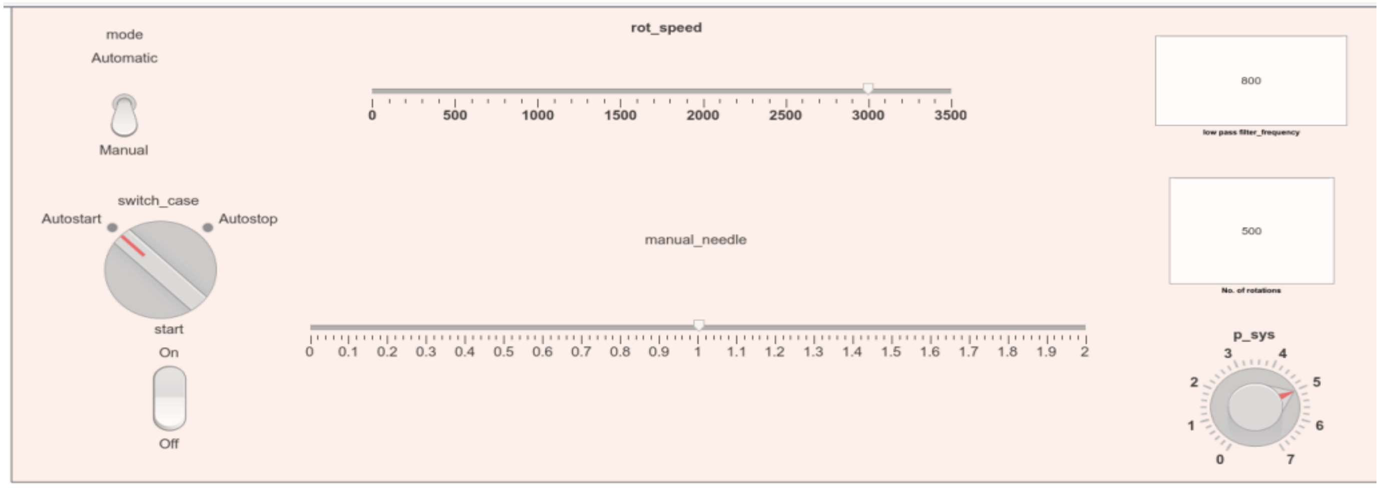


Figure 52: Input Parameters

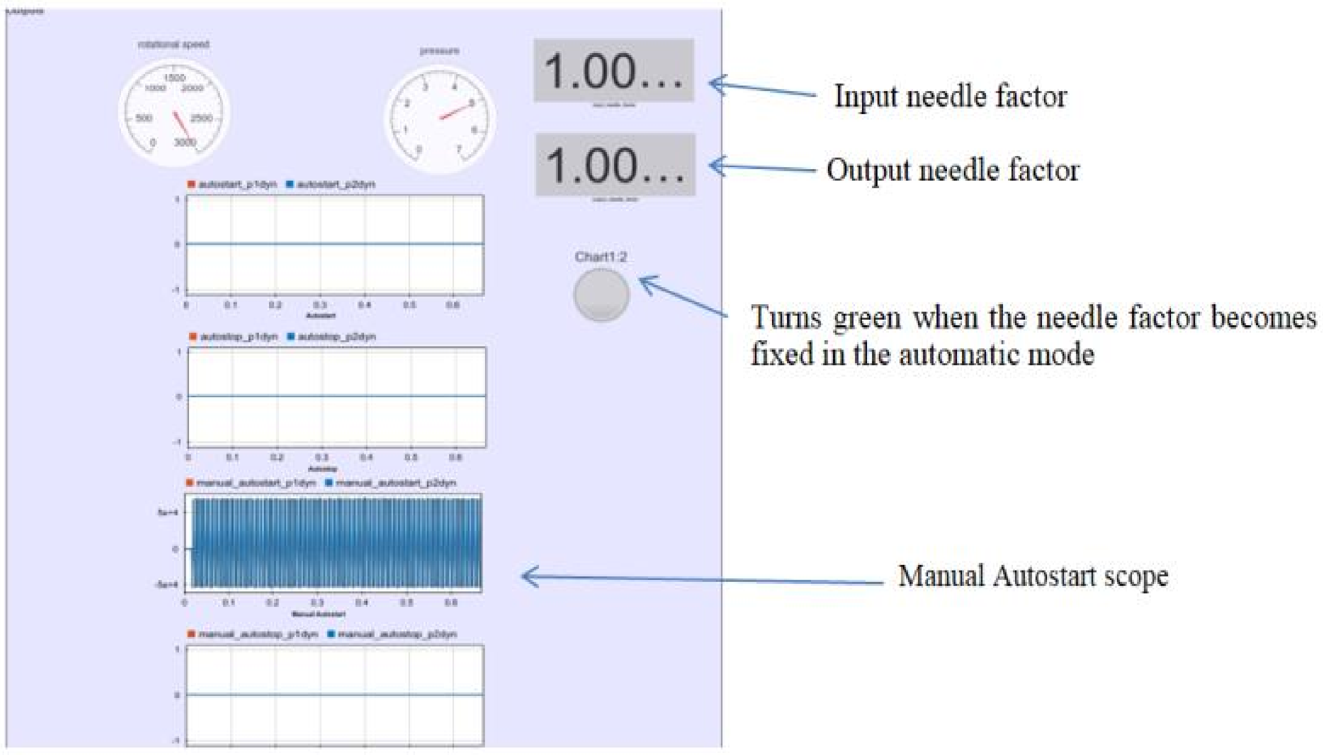


Figure 53: Outputs Window

7.5 Results

7.5.1 EA888 pump

Pressure: 3 bar

Rotational speed: 500 rpm

Needle factor (unitless): 0.757

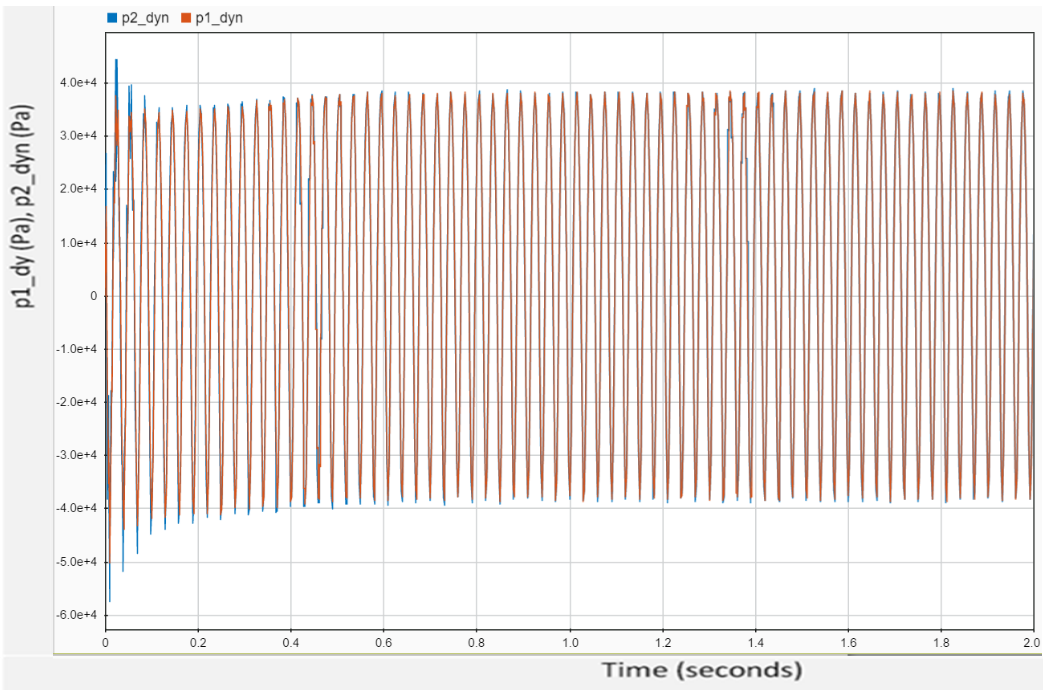


Figure 54: Pressure (Pa)-time(s) curve of the p1 dynamic and p2 dynamic signals

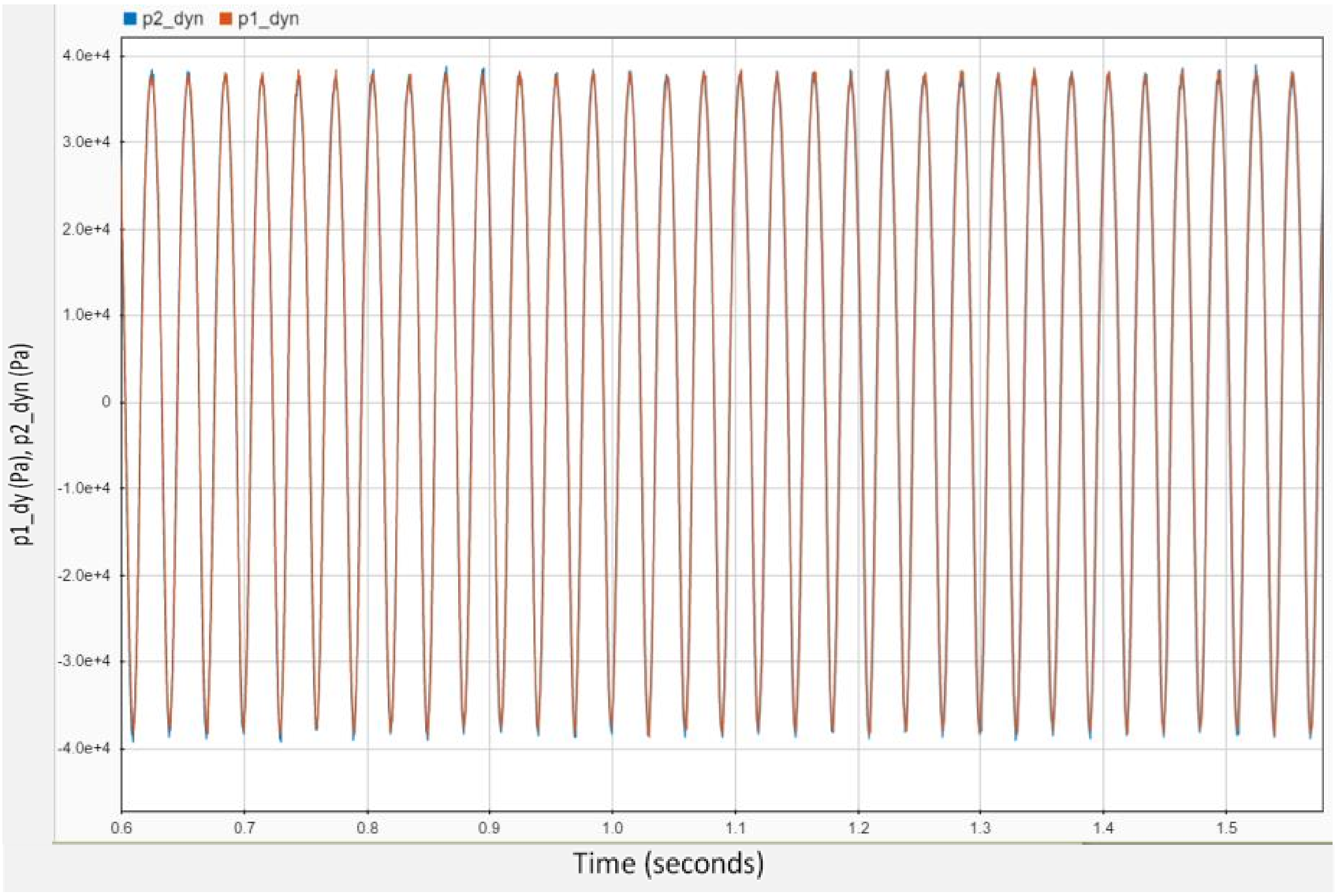


Figure 55: Pressure (Pa)-time(s) curve of the p1 dynamic and p2 dynamic signals (zoomed version)

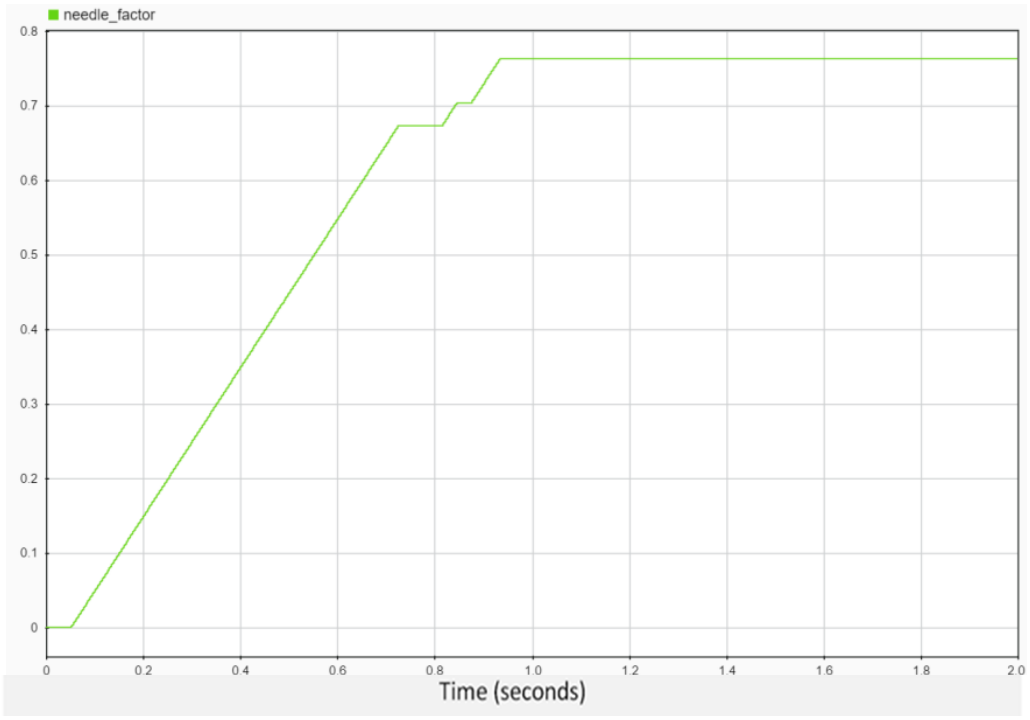


Figure 56: Needle Factor-time (s) graph

Pressure: 3 bar

Rotational speed: 1000 rpm

Needle factor (unitless): 1.02

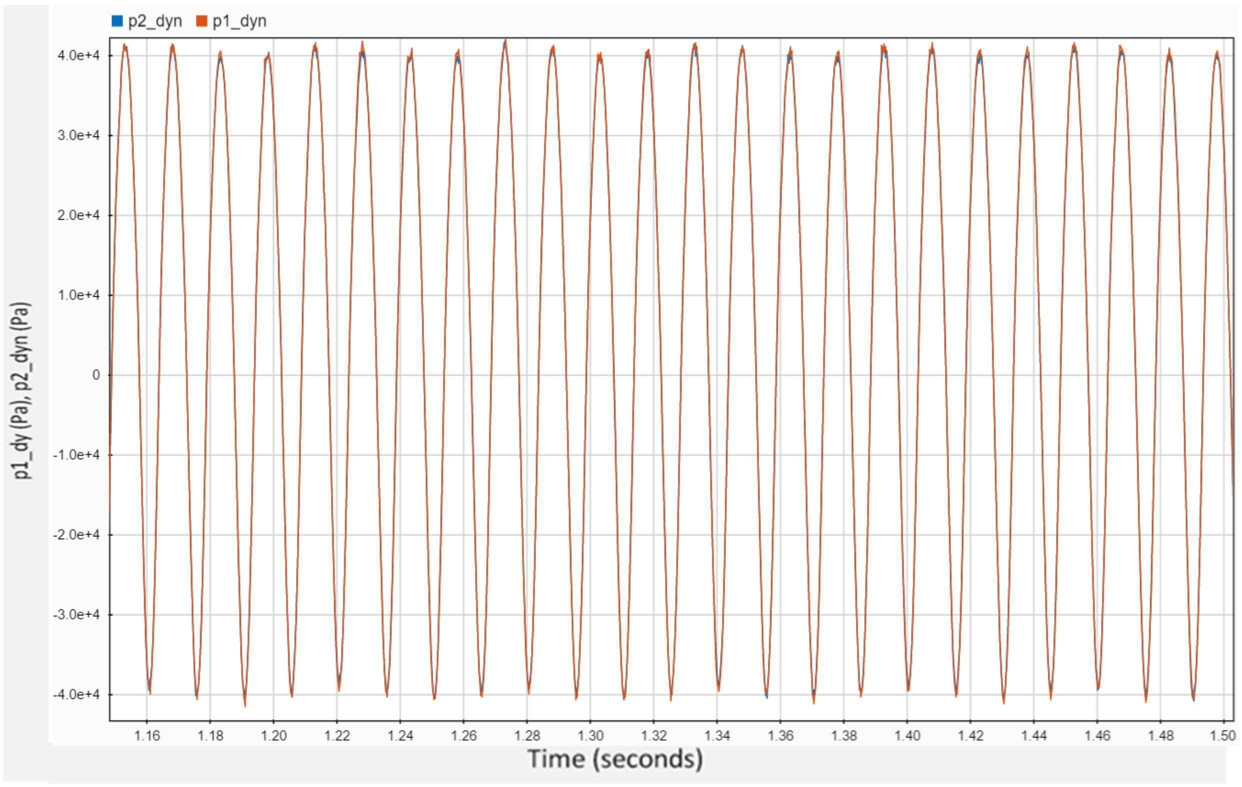


Figure 57: Pressure (Pa)-time(s) curve of the p1 dynamic and p2 dynamic signals

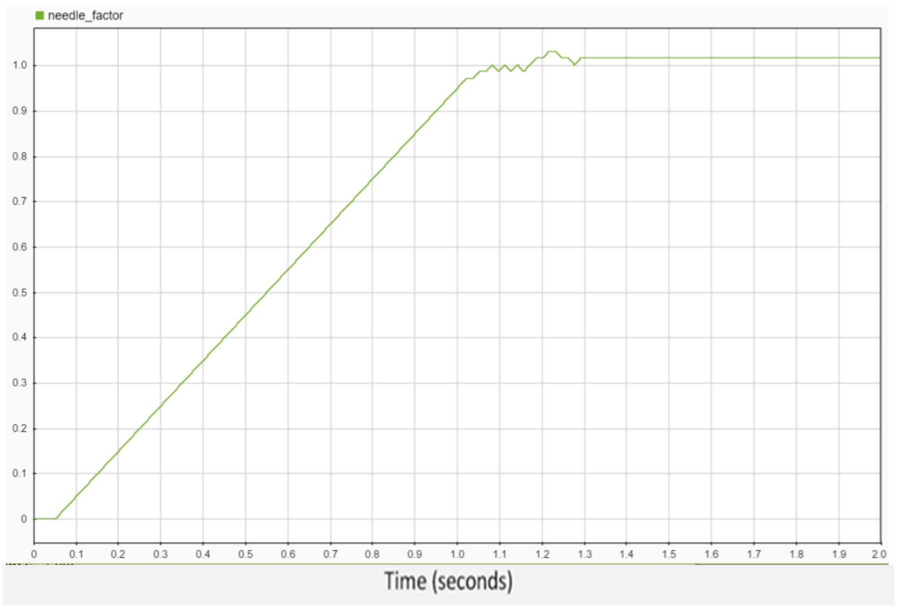


Figure 58: Needle Factor-time (s) graph

Pressure: 3 bar

Rotational speed: 2500 rpm

Needle factor (unitless): 1.01

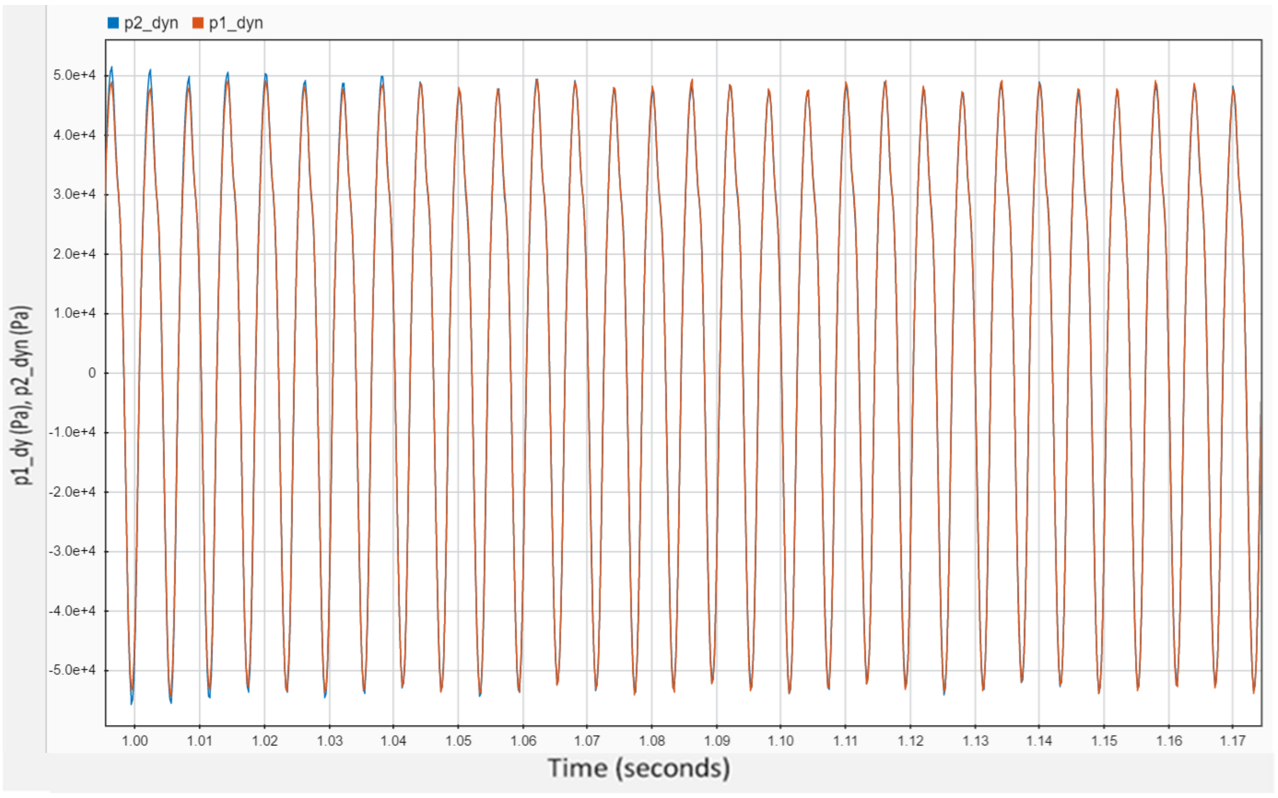


Figure 59: Pressure (Pa)-time(s) curve of the p1 dynamic and p2 dynamic signals

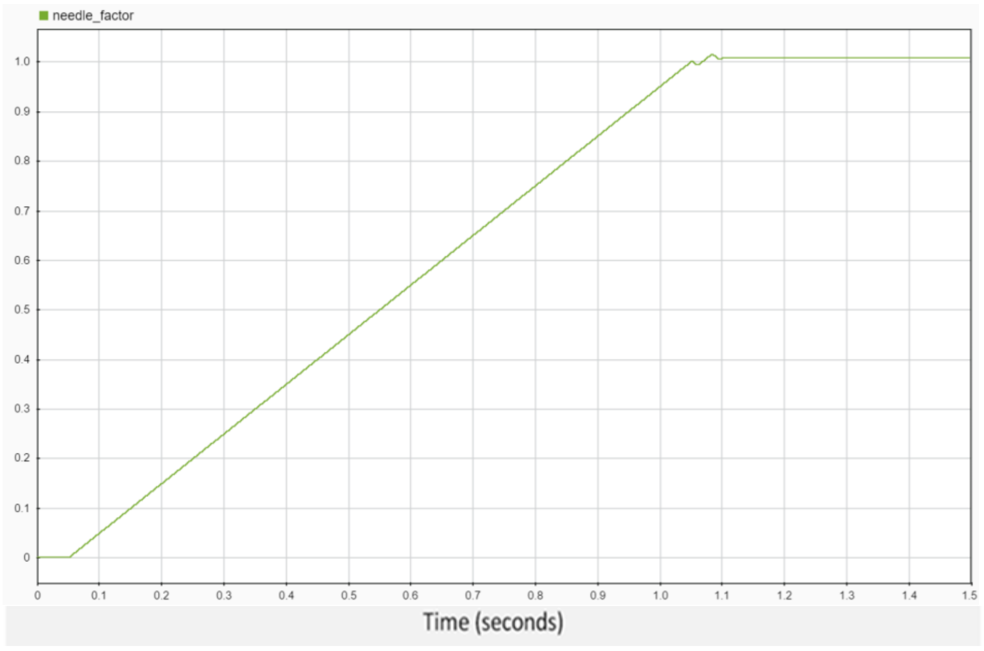


Figure 60: Needle Factor-time (s) graph

Pressure: 3 bar

Rotational speed: 3500 rpm

Needle factor (unitless): 1.01

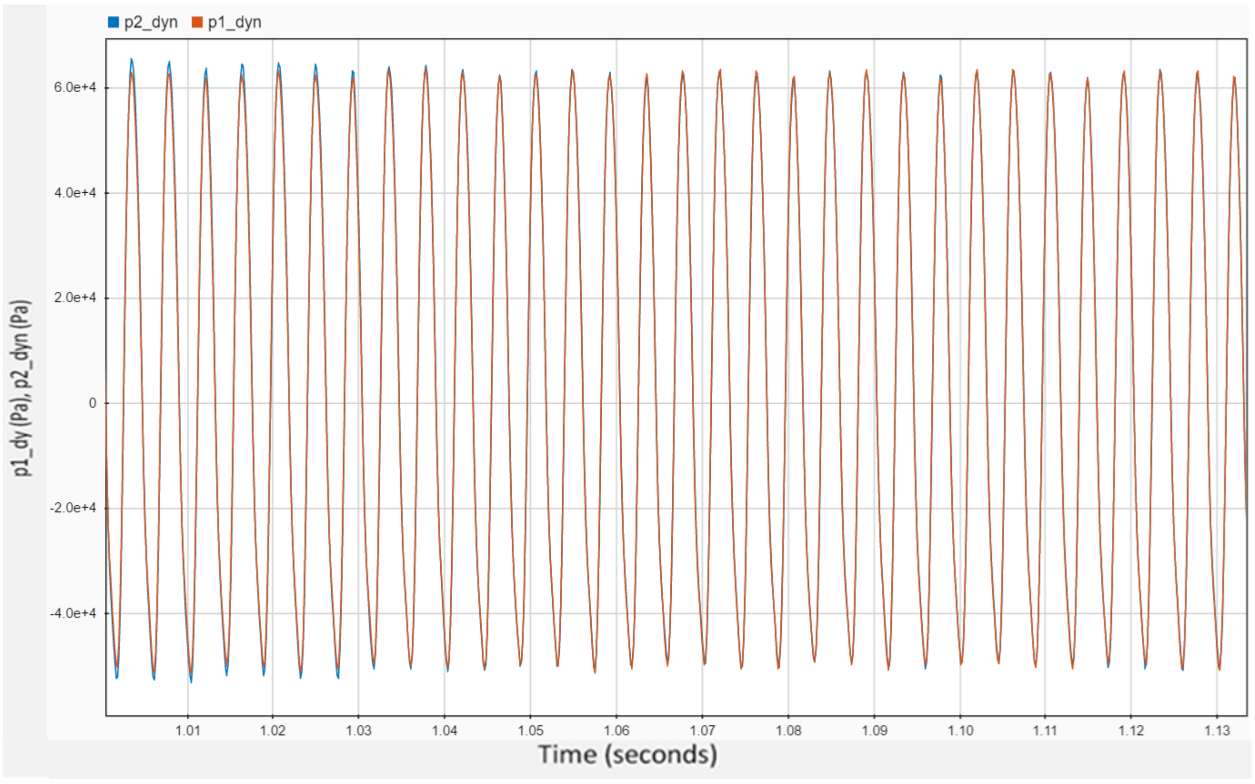


Figure 61: Pressure (Pa)-time(s) curve of the p1 dynamic and p2 dynamic signals

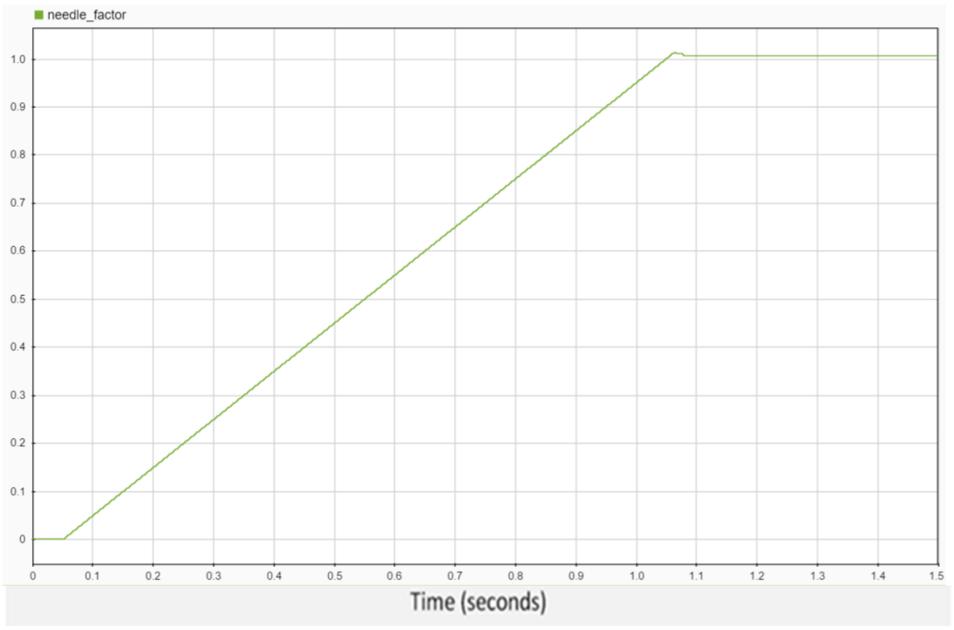


Figure 62: Needle Factor-time (s) graph

Pressure: 5 bar

Rotational speed: 500 rpm

Needle factor (unitless): 0.964

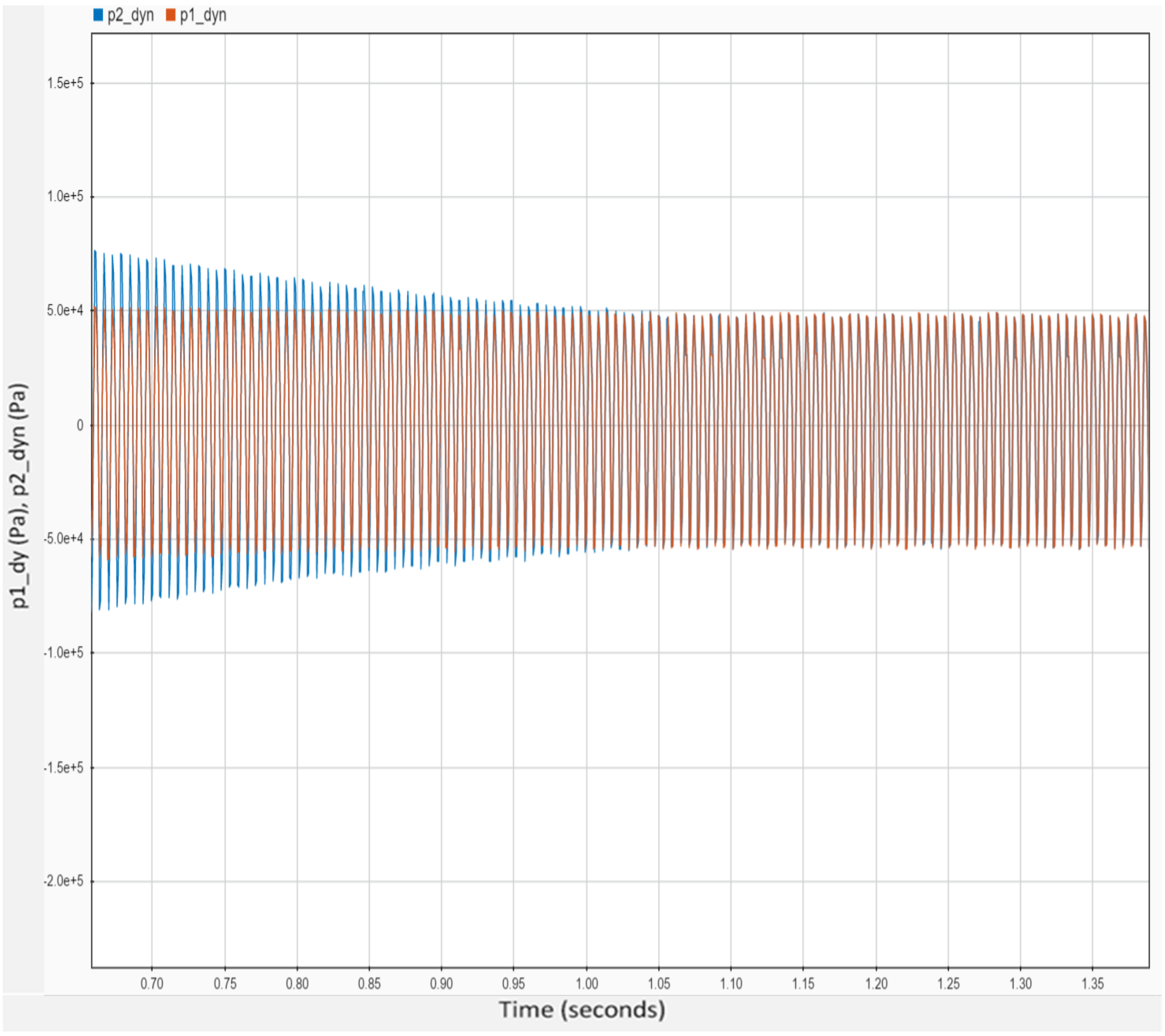


Figure 63: Pressure (Pa)-time(s) curve of the p1 dynamic and p2 dynamic signals

Pressure: 5 bar

Rotational speed: 2500 rpm

Needle factor (unitless): 1.01

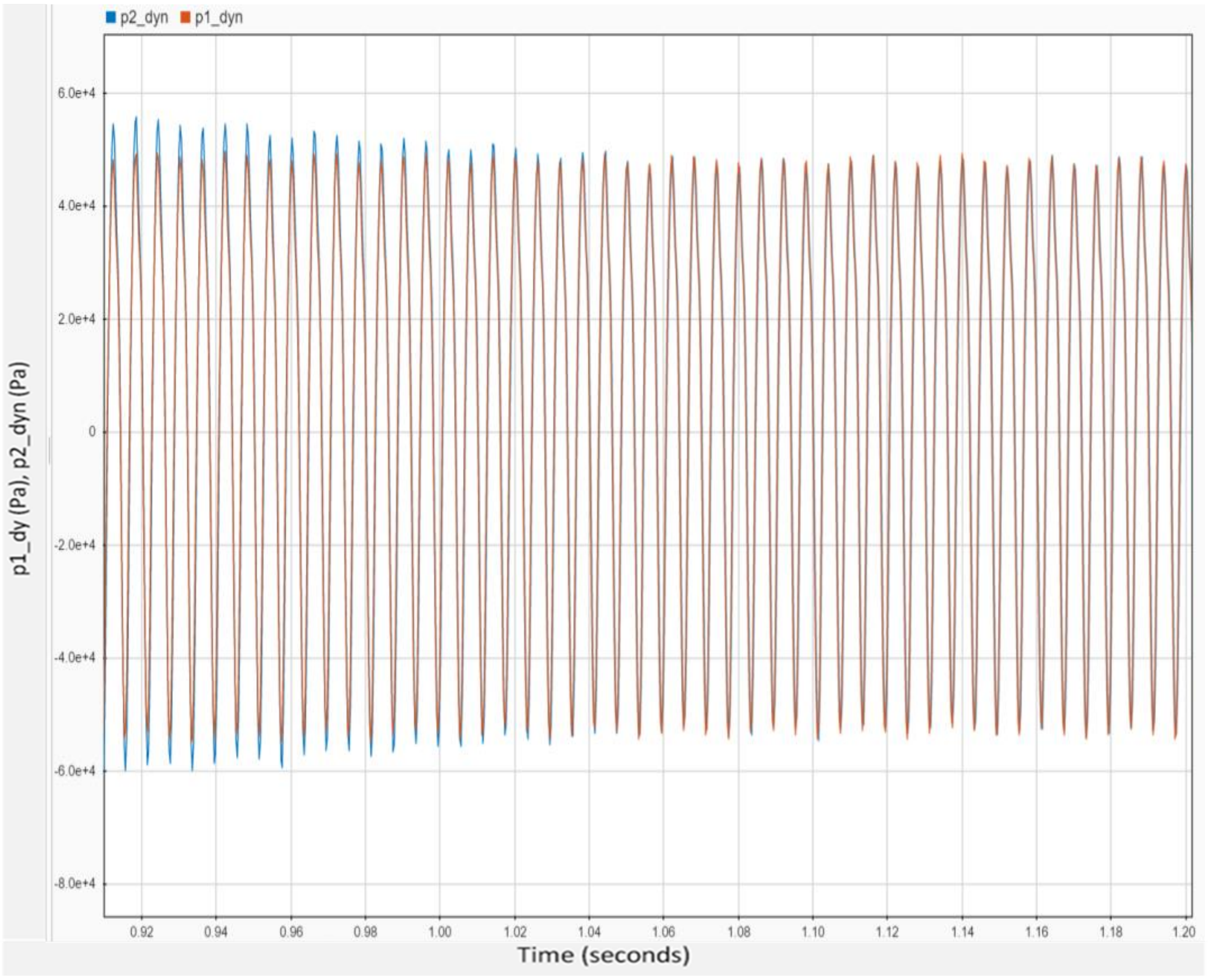


Figure 64: Pressure (Pa)-time(s) curve of the p1 dynamic and p2 dynamic signals

Pressure: 5 bar

Rotational speed: 3500 rpm

Needle factor (unitless): 1.01

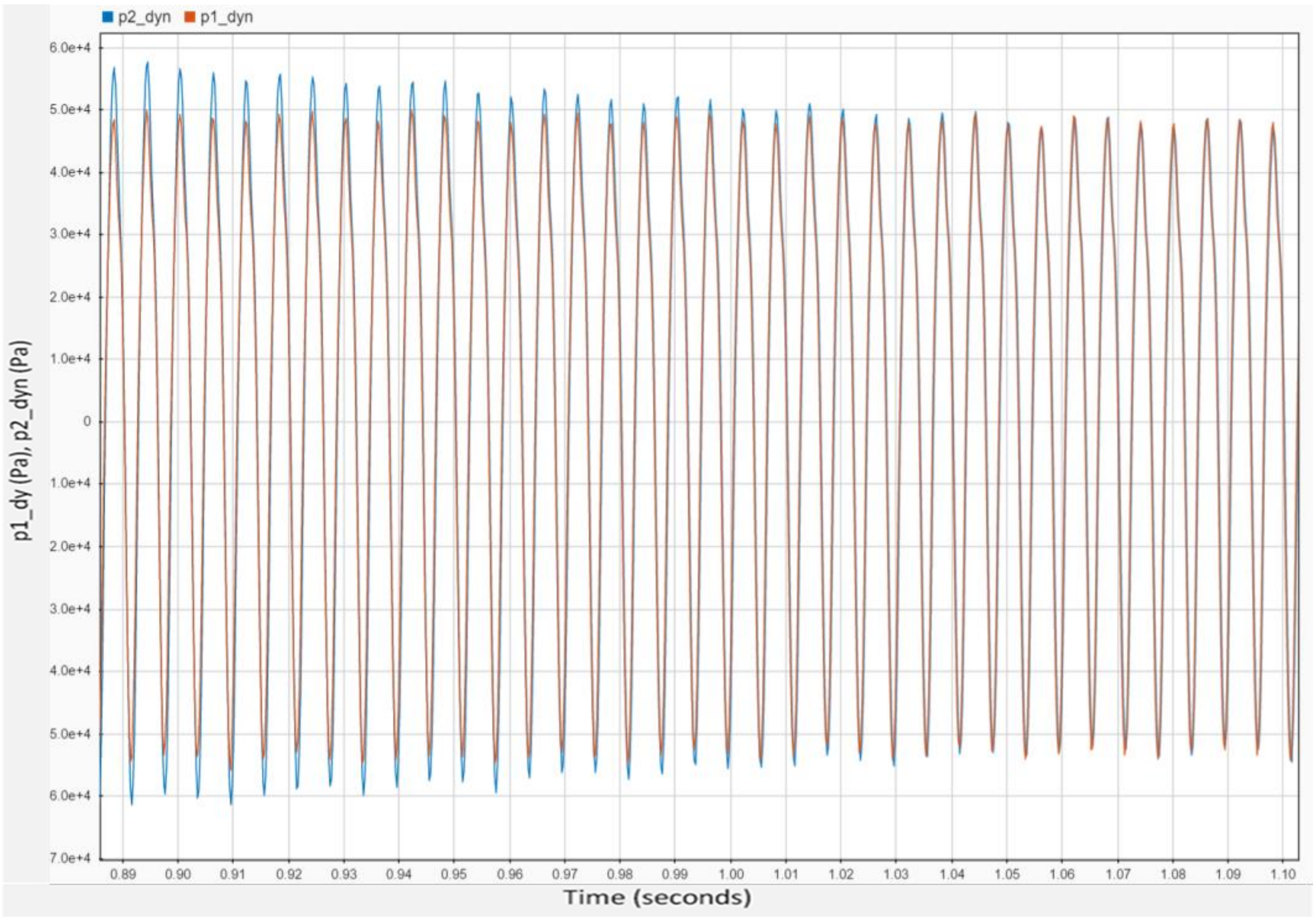


Figure 65: Pressure (Pa)-time(s) curve of the p1 dynamic and p2 dynamic signals

Pressure: 7 bar

Rotational speed: 500 rpm

Needle factor (unitless): 0.854

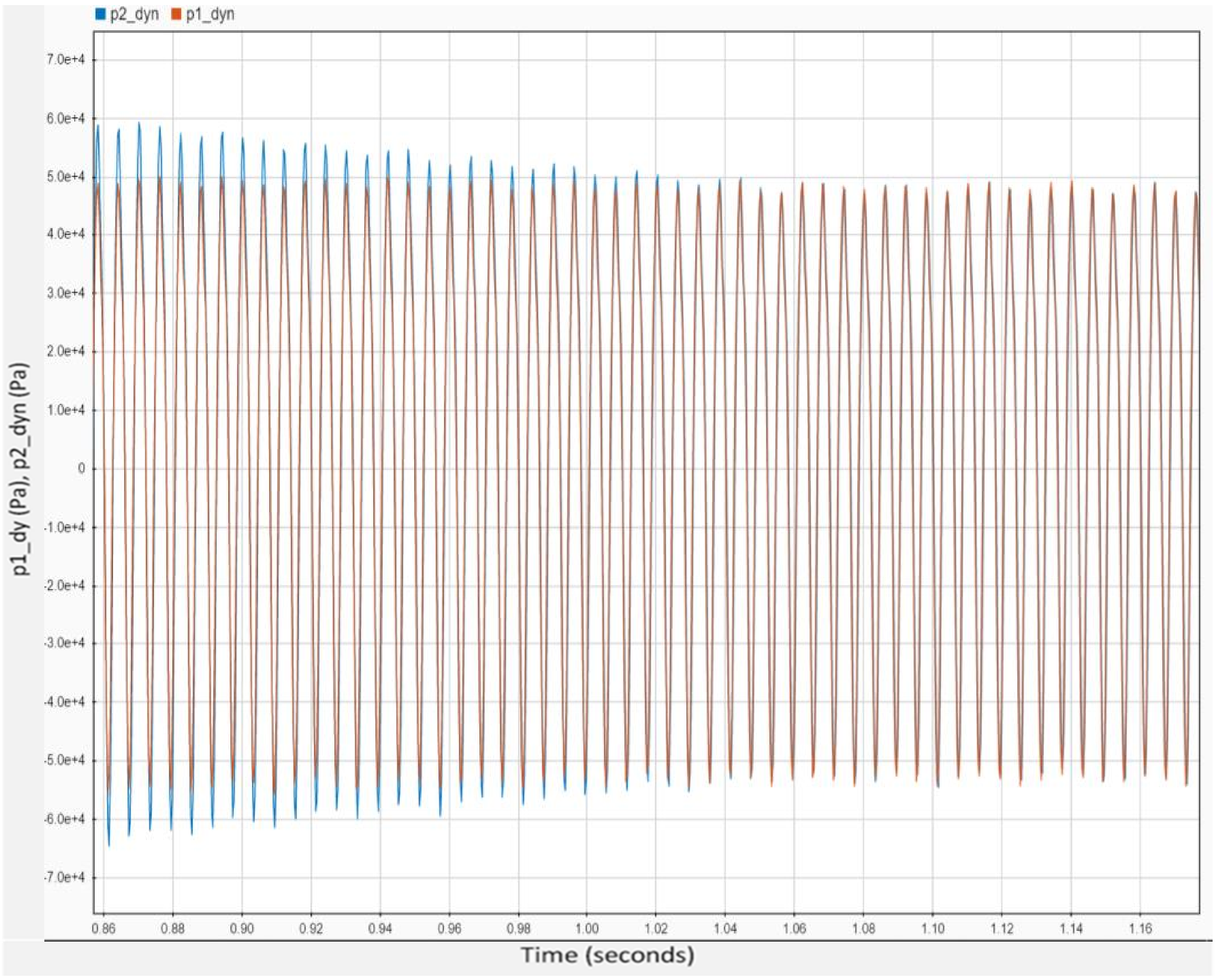


Figure 66: Pressure (Pa)-time(s) curve of the p1 dynamic and p2 dynamic signals

Pressure: 7 bar

Rotational speed: 2500 rpm

Needle factor (unitless): 1.01

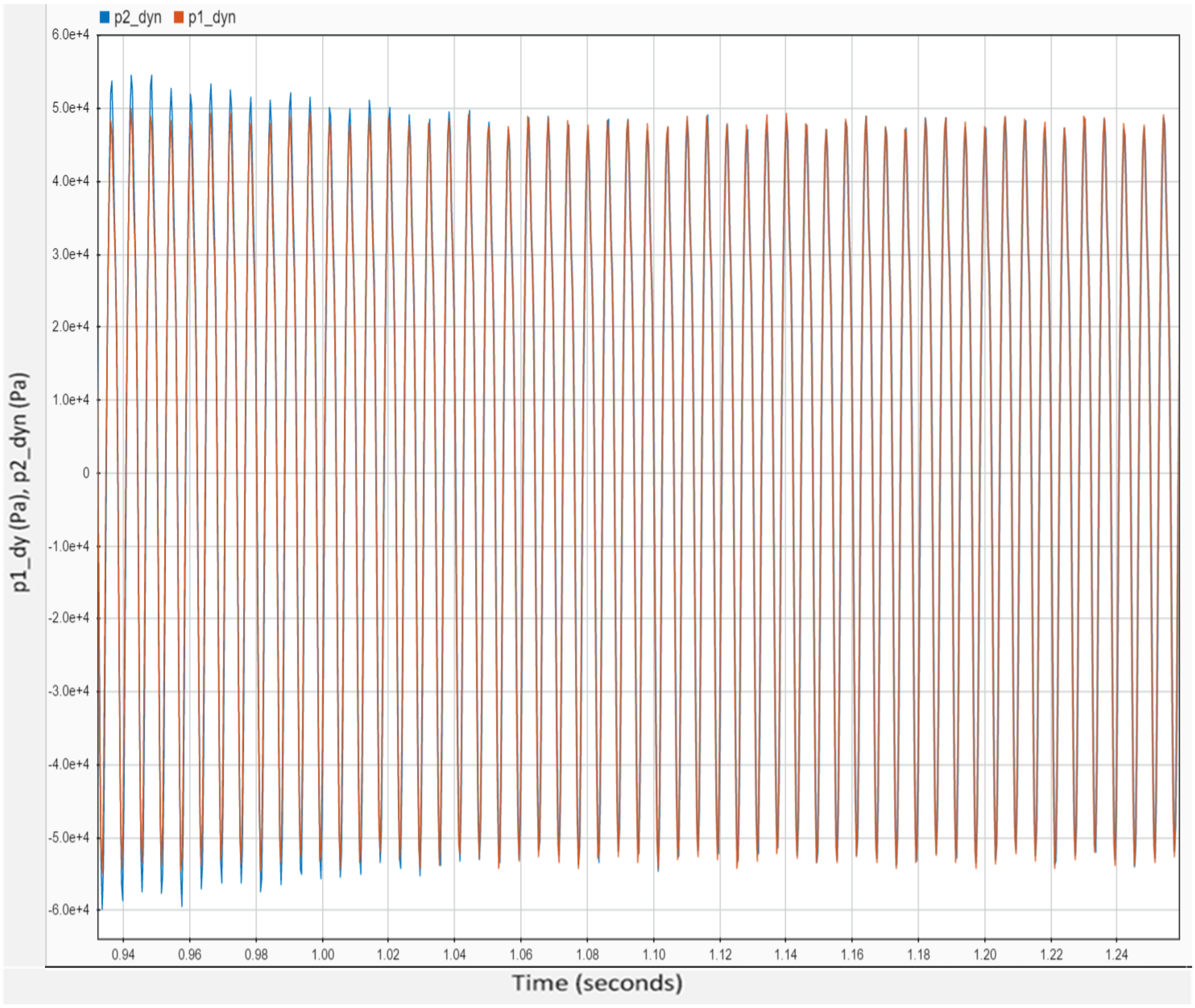


Figure 67: Pressure (Pa)-time(s) curve of the p1 dynamic and p2 dynamic signals

Pressure: 7 bar

Rotational speed: 3500 rpm

Needle factor (unitless): 1.01

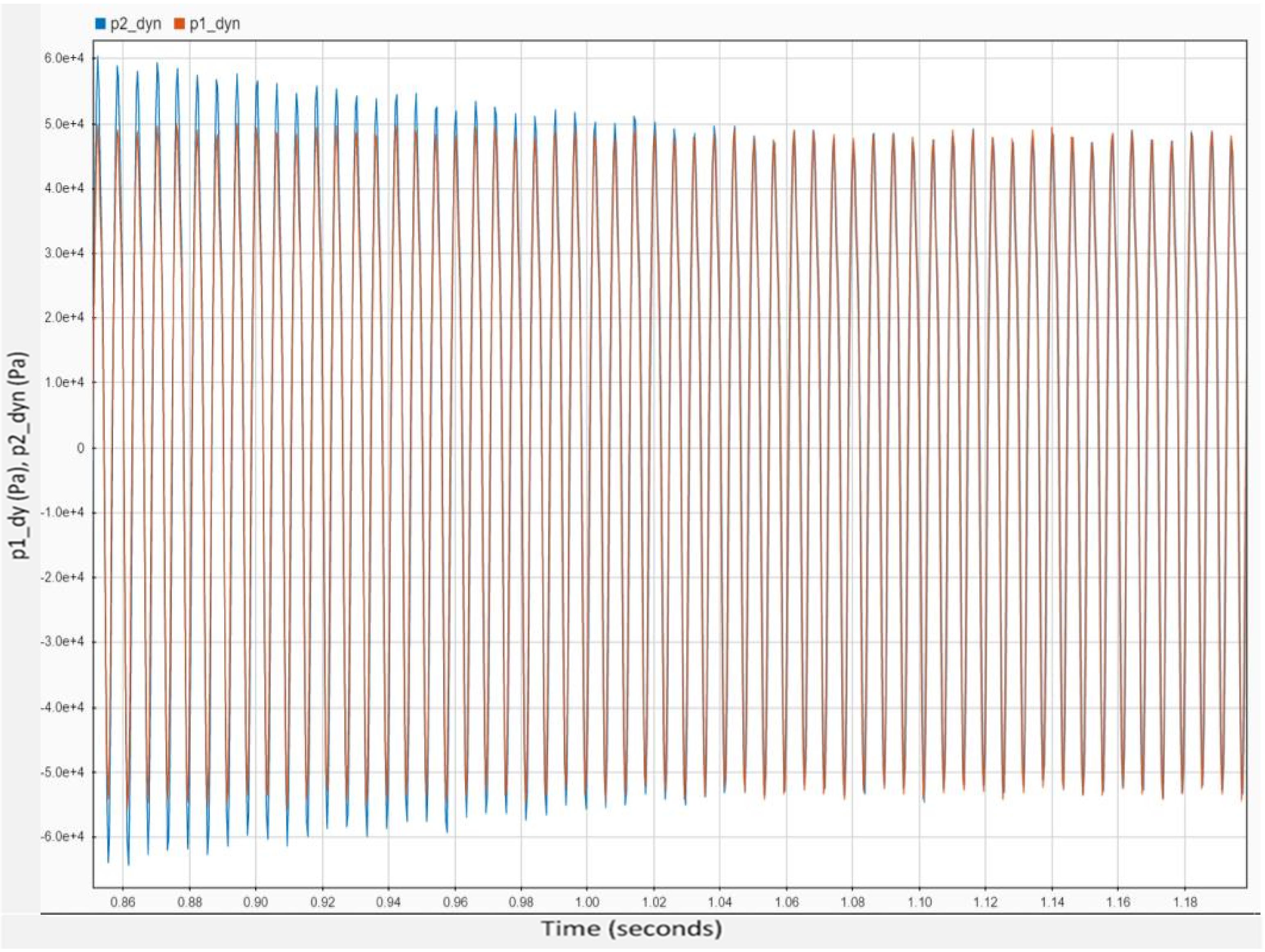


Figure 68: Pressure (Pa)-time(s) curve of the p1 dynamic and p2 dynamic signals

7.5.2 Gen3EVO pump

Pressure: 3 bar

Rotational speed: 1000 rpm

Needle Factor (unitless): 1.01

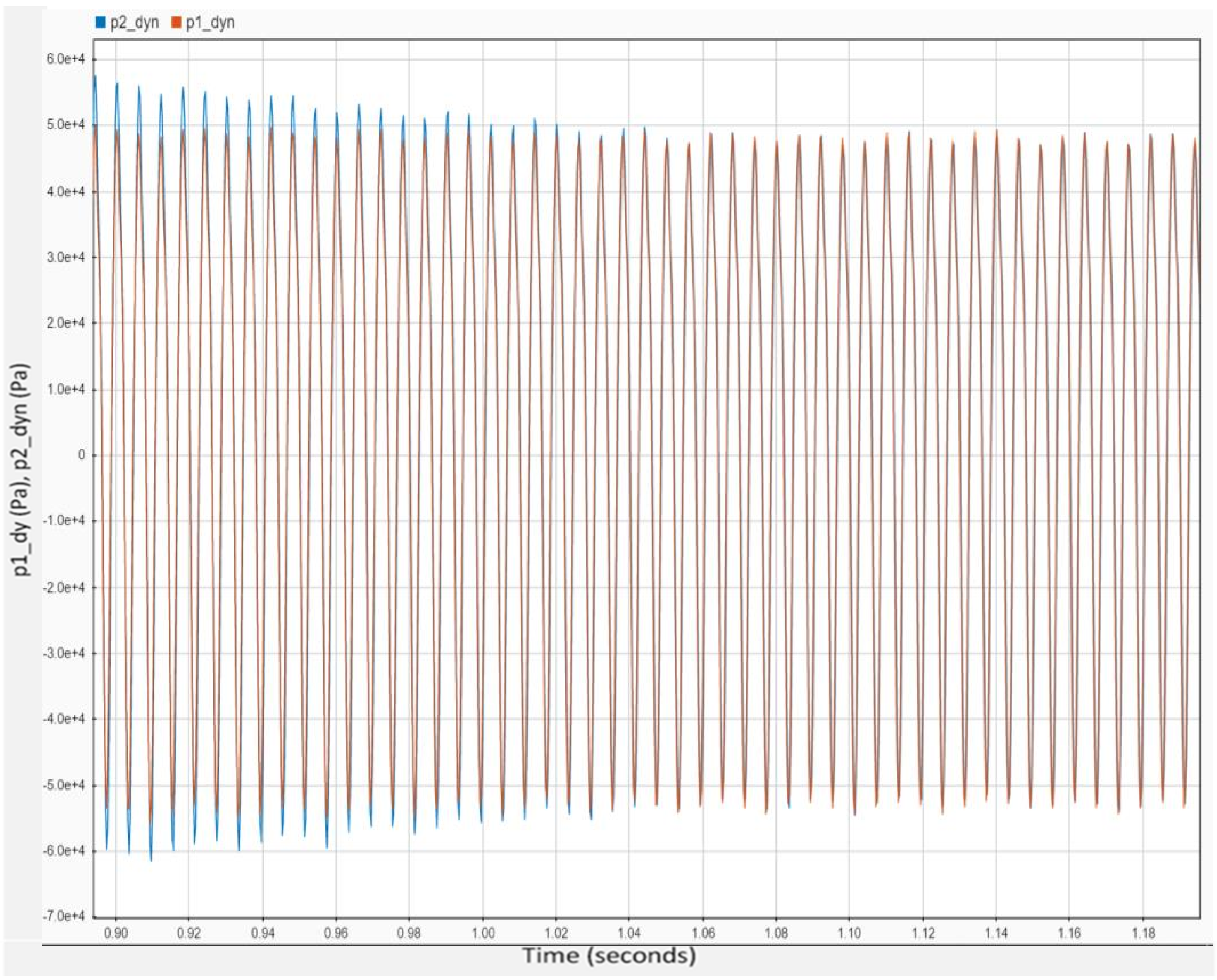


Figure 69: Pressure (Pa)-time(s) curve of the p1 dynamic and p2 dynamic signals

Pressure: 3 bar

Rotational speed: 3500 rpm

Needle Factor (unitless): 1.01

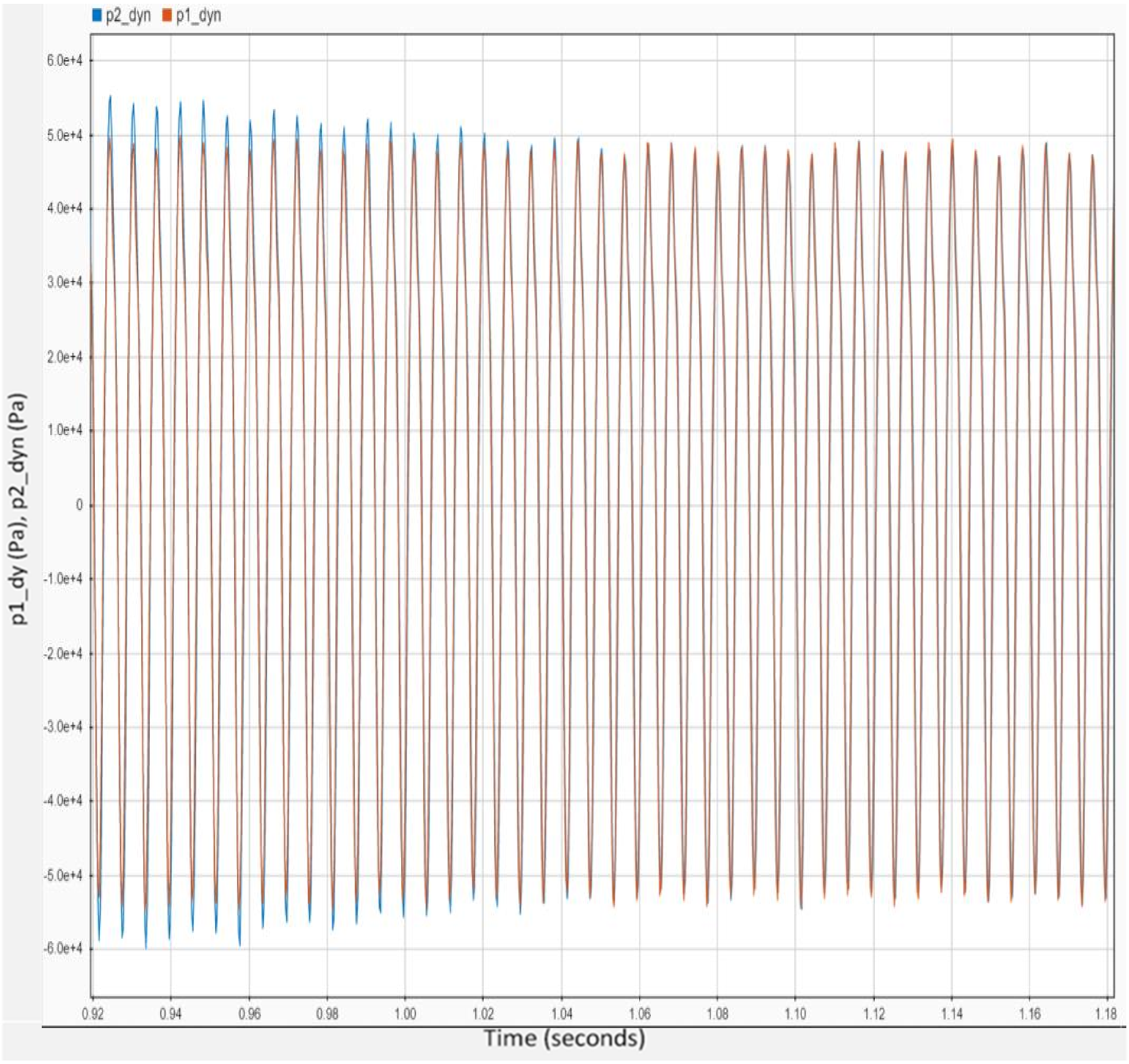


Figure 70: Pressure (Pa)-time(s) curve of the p1 dynamic and p2 dynamic signals

Pressure: 7 bar

Rotational speed: 1000 rpm

Needle Factor (unitless): 0.948

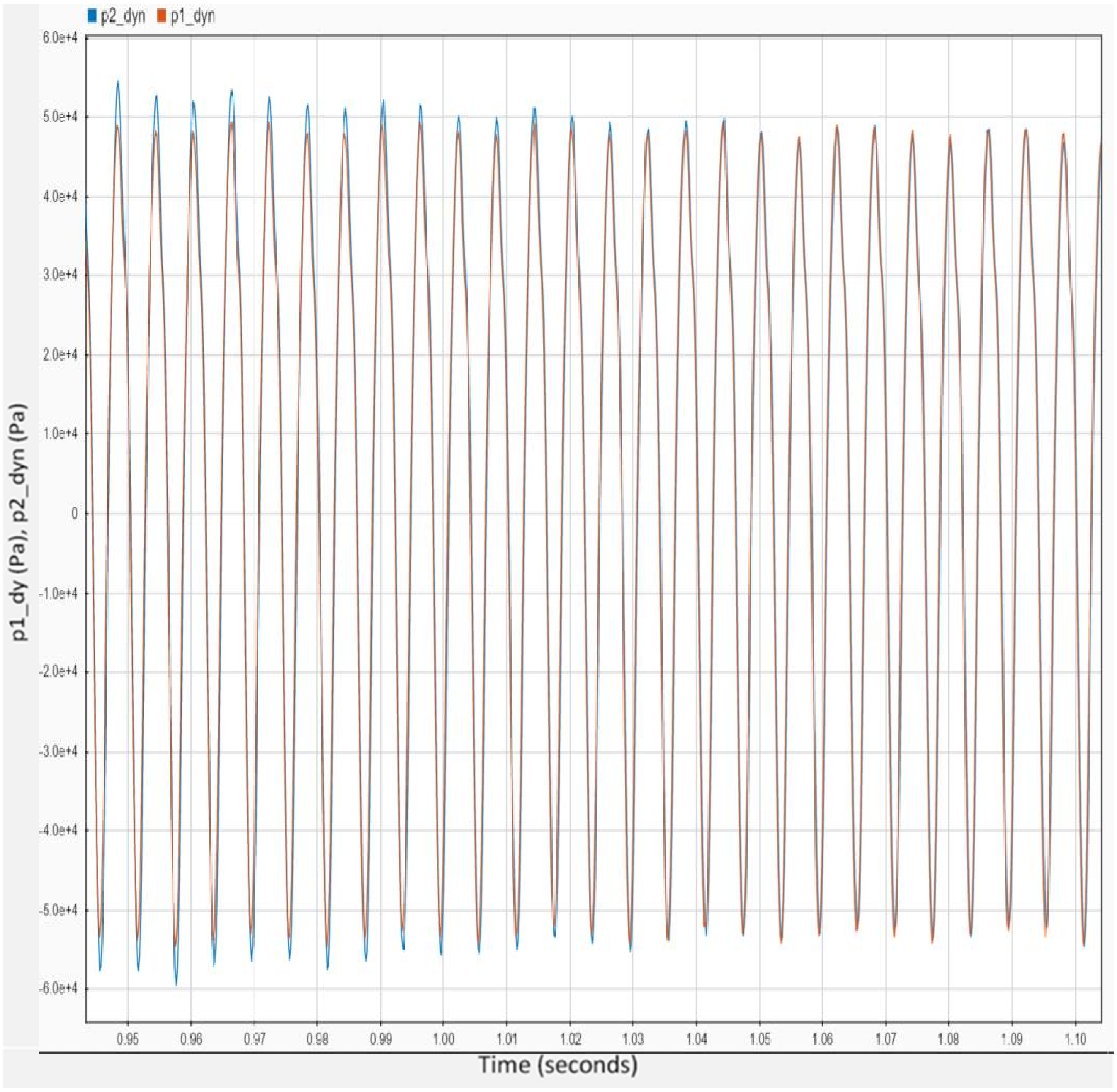


Figure 71: Pressure (Pa)-time(s) curve of the p1 dynamic and p2 dynamic signals

Pressure: 7 bar

Rotational speed: 3500 rpm

Needle Factor (unitless): 1.01

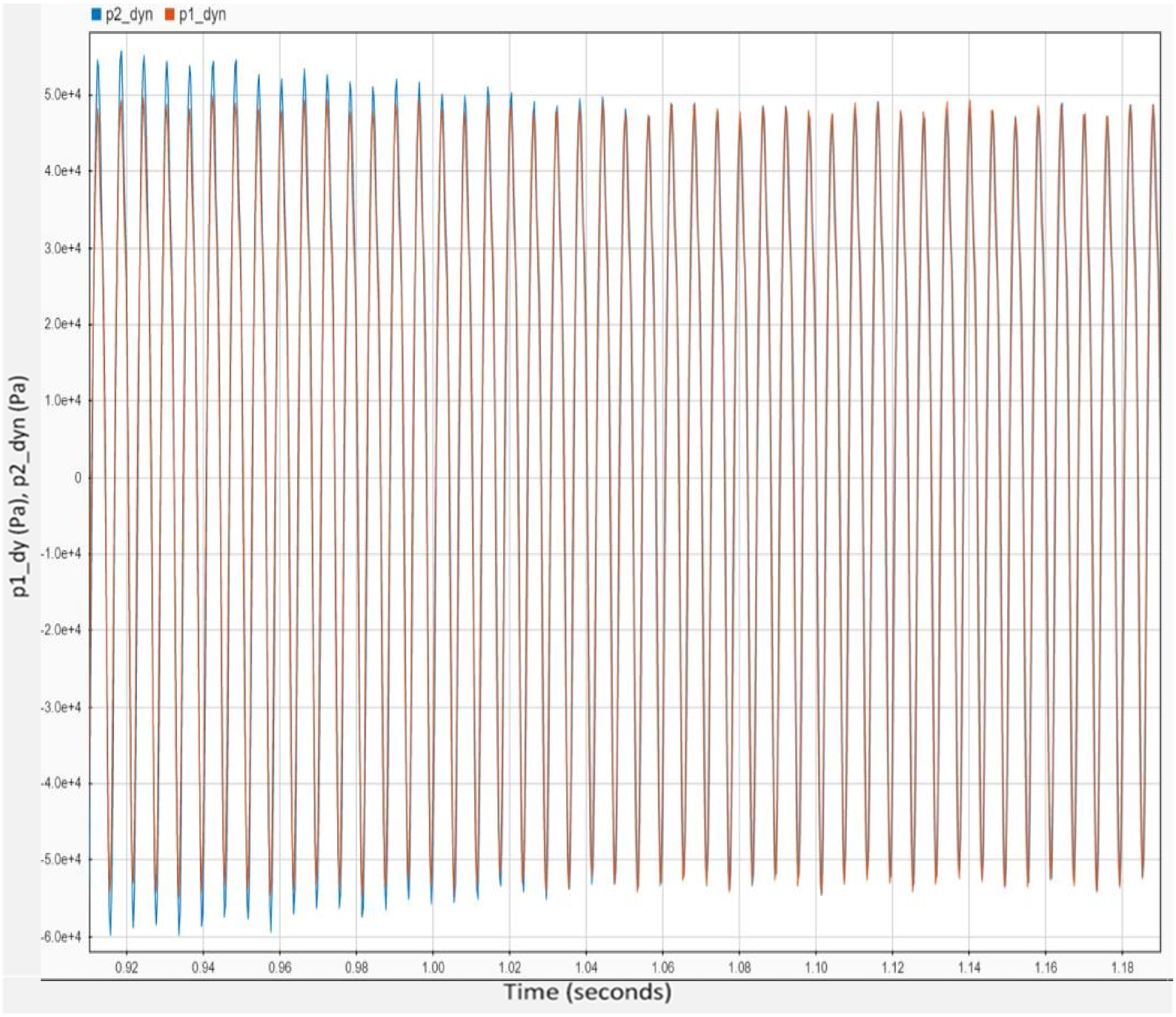


Figure 72: Pressure (Pa)-time(s) curve of the p1 dynamic and p2 dynamic signals

8 Previous Controller Algorithm

Summary: The current controller algorithm was developed after having run numerous simulations on previous controller algorithms and having done a lot of modifications in order to arrive to a robust algorithm that doesn't depend on any external interference which will be explained in this chapter. Robustness is the ability of an algorithm to cope with errors during execution and cope with erroneous input which was missing in this algorithm because some parameters needed to be changed at every new measurement in order to get a proper outcome or result. Thus, it is important to shed lights on the inputs used to feed this algorithm with data and on the logic this algorithm used to process this data.

8.1 MATLAB Function

In the current algorithm, when a set of conditions are satisfied, a variable "y", which is given an initial value of "0", becomes equal to "1". This process takes place through a simple "if statement". Along with the MATLAB function there is a counter which counts the number of times the variable "y" becomes "1" (**Figure 39**). This function was important due to the fact that sometimes the algorithm fixed an erroneous value of the needle factor because for an instance of time, all the conditions were satisfied but not maintained. This is why it was of great significance to introduce this variable in order to guarantee that all the conditions were satisfied for a specific number of times before the needle factor is fixed. However, this function was inexistent in the previous algorithm which made it susceptible to faulty results.

8.2 Stateflow Chart

As seen in **Figure 73**, the inputs to the Stateflow chart that contains the main logic of the controller algorithm are:

- **p**: the difference between the maxima of p1_dyn & p2_dyn signals.
- **p1**: the data from *p1_dyn* signal
- **clck**: which represents the difference between the x-intercepts of the 2 dynamic pressure signals p1_dyn & p2_dyn (i.e.: x-intercept (*p1_dyn*) - x-intercept (*p2_dyn*))

The only **output** is the needle factor *n*.

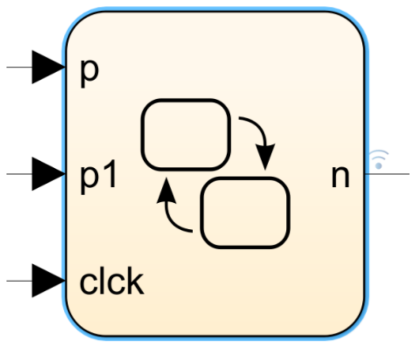


Figure 73: The stateflow chart which holds the main logic of the controller algorithm

An inside look into the main logic

As seen in **Figure 74**:

- The first two "initial" blocks are used to delay the start of the calculations in order to give some time, even though it might be relatively small, for the system to stabilize & deliver the expected results.
- The third "initial" block is the center block or the "decision maker", in which, the controller waits to process the data & move to the suitable block (increase, decrease or keep the value of the needle factor fixed).
- Increasing the needle factor: The conditions to move to this block are when "**p**" which is the difference between the two signals: *p1_dyn* and *p2_dyn* is less than zero and the absolute value of "**clck**" is greater than a certain value. When these conditions are satisfied, the needle factor is incremented by a small value (0.00001) each time.

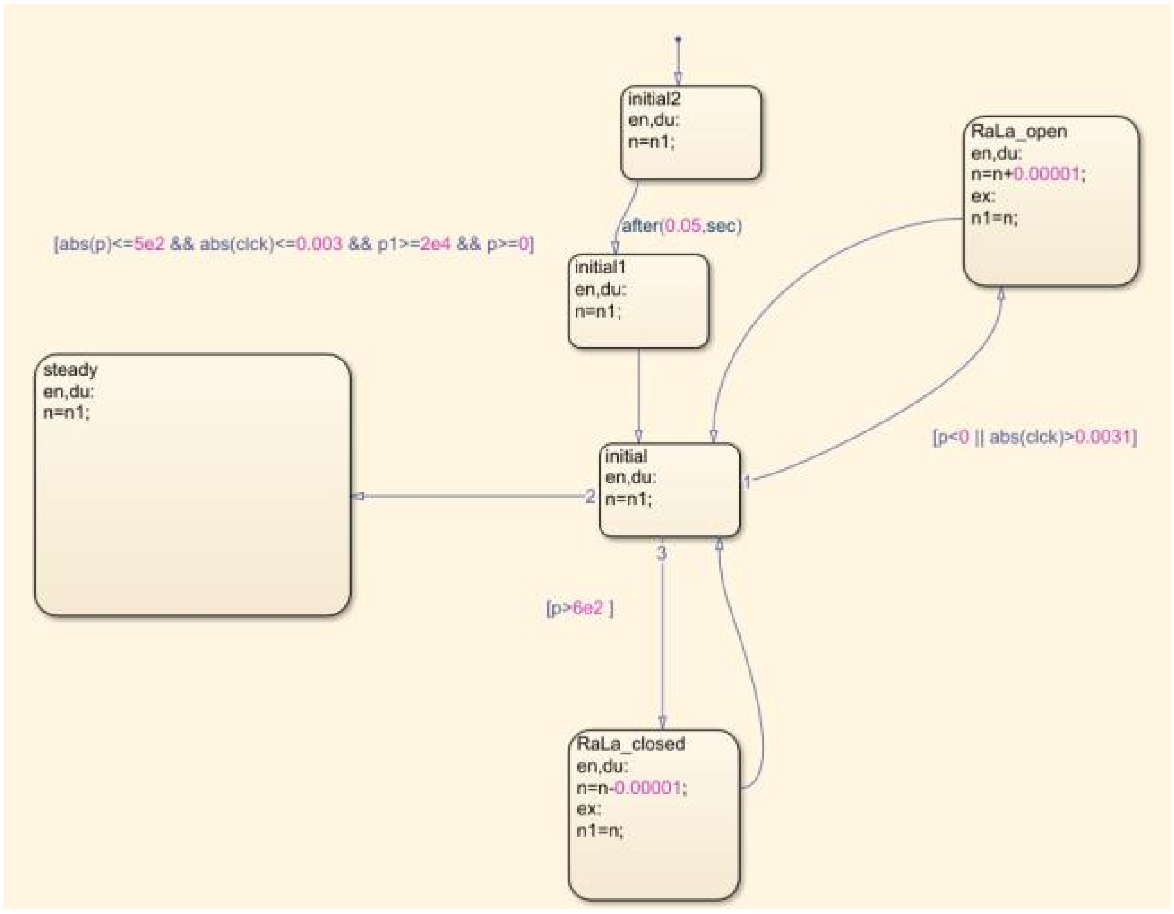


Figure 74: The Stateflow chart's main logic

- Decreasing the needle factor: The conditions to move to this block are when "p" which is the difference between the two signals: p1_dyn and p2_dyn is greater than a certain value. When these conditions are satisfied, the needle factor is incremented by a small value (0.00001) each time.
- Fixing the value of the needle factor (finding the optimal value): The conditions to move to this block are:

  *"***p***" is greater than or equal to zero and the absolute value of "clck" is greater than a certain value. When these conditions are satisfied, the needle factor is incremented by a small value (0.00001) each time.

  *Absolute (***p***) <=certain value

  *Absolute (clck)<=0.0.003

  *p1>=0

Problem: The faulty side in this algorithm or logic is that the results depend on the values that were set as conditions to the parameters inside the transitional phase between the blocks. As a result to this error, the accuracy of the results was affected heavily and these values needed to be changed and calibrated before and after each simulation.

9 EtherLab

Summary: *Beckhoff* is a German company that created a global standard for automation with the launch of PC-based control technology in the mid-80s. On the software side, the Twin-CAT (The Windows Control and Automation Technology) automation suite forms the core of the control system. The TwinCAT software system turns almost any PC-based system into a real-time control with multiple PLC, NC, CNC and/or robotics runtime systems. The RaLa was controlled and driven using this system where *EtherCAT,* a real-time Ethernet network, was used to communicate with the motor and the rest of the electrical and electronic devices. These devices such as the sensors and motors form the backbone of the test-rig by acquiring and exchanging data between the components from one side and the algorithm or the user from the other side through a nice and neat GUI (Graphical User Interface). The terminal box that is considered the connecting point between all the components is formed of Beckhoff terminals connected to each other through a coupler that exchanges data with the computer. At this point our DSHplus model has served its purpose by providing us with detailed simulations in order to arrive to a point where the controller algorithm becomes a robust algorithm that satisfies all the requirements. However, this model has to be replaced with blocks that merely represent the sensors and the other electrical and electronic devices in this circuit, so this step was accomplished using EtherLab blocks.

9.1 Introduction

DSHplus is powerful software that provides the hydraulic model with blocks and parameters that emulate the behavior of the real hydraulic circuit with all its components. It provides us with accurate simulations that reproduce the physical and hydraulic restrictions in the circuit through the available lookup tables, parameters, and units of measurement and blocks that represent the hydraulic components. During the process of developing a robust control algorithm to adjust the position of the needle in the RaLa device through the "Faulhaber motor", data exchange took place through the input and output blocks in the DSHplus model. However, when implementing this algorithm on the real test-rig these blocks, the latter blocks need to be replaced by a certain kind of code that represents the terminals that receive and send the data on the test-rig. The EtherLab blocks in Simulink hold this piece of code which by its turn contains the data defining these terminals making them familiar to the framework introduced on the test-rig.

9.2 What is Beckhoff and what part does its software play in the test-rig?

"Beckhoff implements open automation systems based on PC Control technology. The product range covers Industrial PCs, I/O and Fieldbus Components, Drive Technology and automation software. Products that can be used as separate components or integrated into a complete and seamless control system are available for all industries. The Beckhoff "New Automation Technology" philosophy represents universal and open control and automation solutions that are used worldwide in a wide variety of different applications, ranging from CNC-controlled machine tools to intelligent building automation. It provides an extensive range of fieldbus components for all common I/O and fieldbus systems. The wide choice of I/O components means that the bus system best suited to the particular application can be chosen."(6)

9.3 What is EtherCAT?

"EtherCAT (Ethernet for Control Automation Technology) is an Ethernet-based fieldbus system, invented by Beckhoff Automation. It is the open real-time Ethernet network originally developed by Beckhoff. EtherCAT sets new standards for real-time performance and topology flexibility. .The protocol is standardized in IEC 61158 and is suitable for both hard and soft real-time computing requirements in automation technology. The goal during development of EtherCAT was to apply Ethernet for automation applications requiring short data update times (also called cycle times; ≤ 100 µs) with low communication jitter (for precise synchronization purposes; ≤ 1 µs) and reduced hardware costs." (7)

Automation equipment vendors can utilize *EtherCAT* on their own device implementations to improve performance and flexibility. End users and automation system designers can also implement their own EtherCAT compliant devices for specific needs which was done on the test-rig by connecting the *Faulhaber motor,* which was invented by a German company that isn't a part of Beckhoff, to the terminal box because it is an EtherCAT compliant device. For networks with many drives, I/Os, and other devices, the transmission overhead can be significantly reduced with this approach (**Figure 80**). This efficient network design of EtherCAT makes it ideal for high bandwidth applications like multi-axis servo machine control. Sampling and updating 64 drives and many I/Os devices can be done in less than 250 microseconds.

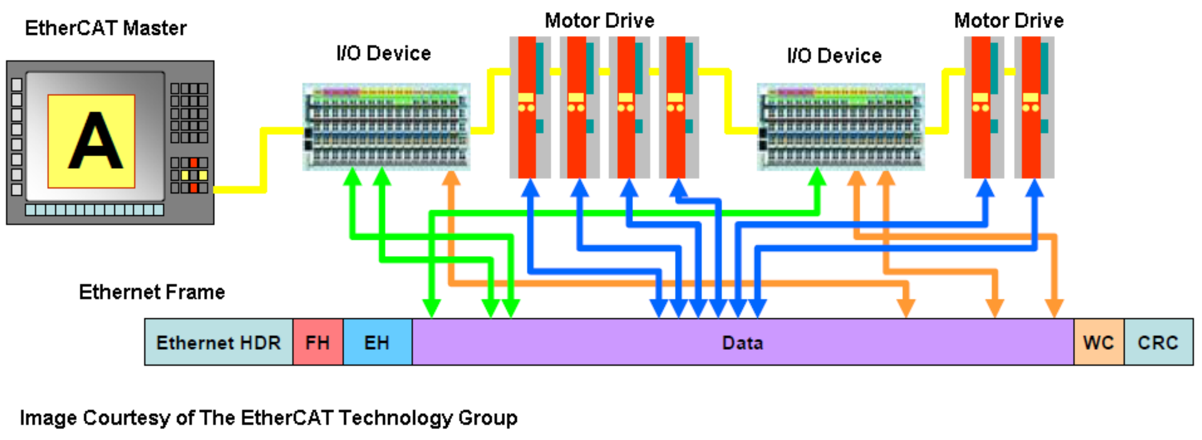


Figure 75: The EtherCAT Master-slave connections

9.4 What is EtherLAB?

“EtherLab is a technology combining hard- and software for test and automation purposes. It is not a product but a technique built from reliable and well known components and also new technical approaches. Basically EtherLab works as a Real Time kernel module attached to the open source operating system Linux communicating with peripherals devices by a special Ethernet technology, known as EtherCAT.

The main features of this powerful technology:

- Open Source
- Hard Real Time
- Simulink/RTW Code Generation
- EtherCAT Block set
- Multi-Client, -User, -Server, -Tasking
- Additional Services
- Industrial Grade I/O Stations
- High Efficiency
- Small Hardware Costs
- Remote Maintenance
- Cost-saving
- Flexibility
- Windows and Linux Frontend” [(8)]

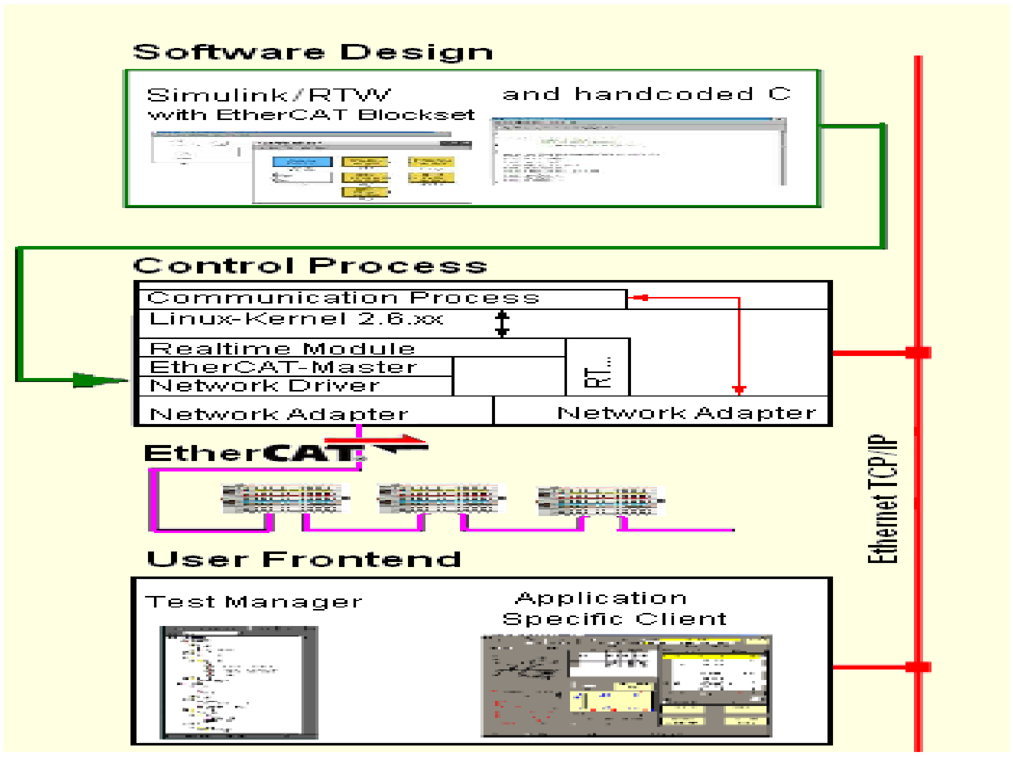


Figure 76: How the communication between the different modules is established [(8)]

9.5 Which EtherLAB blocks were used in the project and what are their properties?

9.5.1 Target Pressure

EL 4132

In order to guarantee that the target pressure is reached, an electro pneumatic regulator (**Figure 79**) maintains constant output pressure in compressed-air systems regardless of variations in the input pressure or output flow.

The dedicated block is responsible of receiving the data from the user via the HMI and passing them to the regulator through a proper PDO. This block represents the corresponding Beckhoff terminal on the terminal box that has the following properties:

- analog output terminal
- generates signals in the range between -10 and +10 V
- resolution of 16 bits
- 2 channels

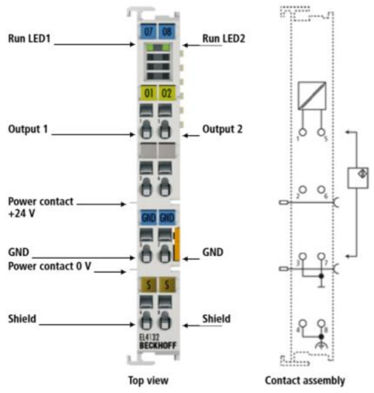


Figure 77: The EL 4132

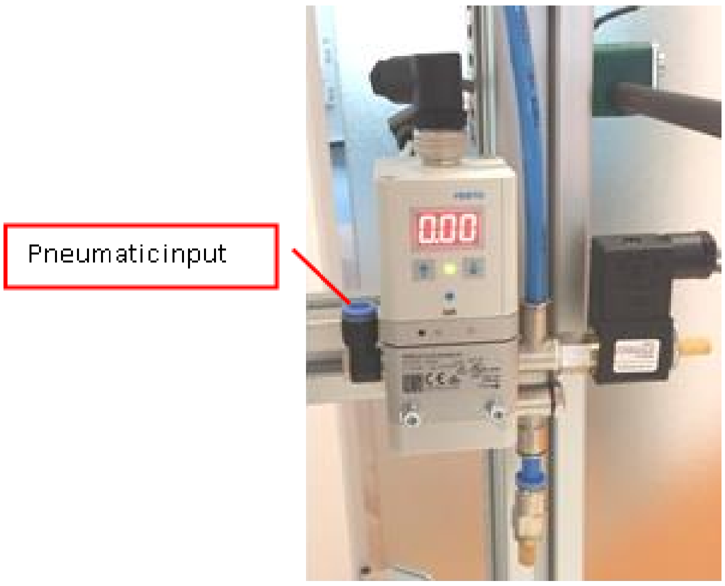


Figure 78: input pneumatic pressure connection

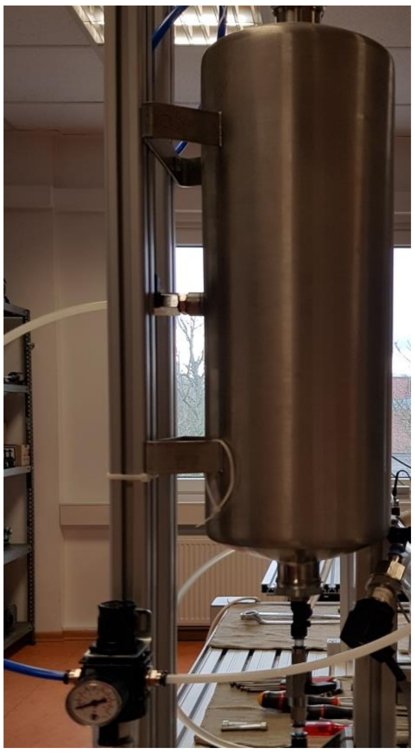


Figure 79: The pressure tank with the electro pneumatic regulator

In the Simulink model, the value of the target pressure (in bar), inserted by the user, is converted to volts by multiplying it by a gain **equal to 1** as 1 bar corresponds to 1 Volt**.**

9.5.2 Rotational speed

EL 4132

Initially, the user inserts the desired rotational speed on the HMI; this value is then passed to the electric motor (**Figure 80**) through the dedicated frequency converter via the same terminal that we used for the target pressure. In the Simulink model, the value is converted from rpm to volts using a gain block of value (1/344) because 344 rpm correspond to 1 volt.

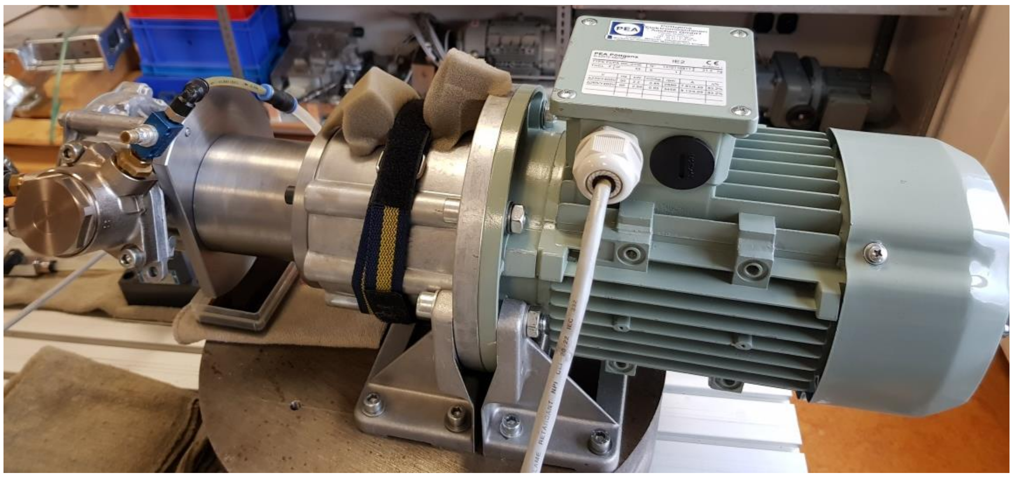


Figure 80: The asynchronous drive motor

9.5.3 Dynamic pressure

EL 3632

In order to measure the dynamic pressure signals *p1_dyn* and *p2_dyn*, a PCB pressure sensor (**Figure**) with a range of 345 kPa was used that communicates with the EL 3632 Beckhoff terminal, which by its turn passes the data to the controller algorithm in order to be processed.

This block (**Figure 83**) represents the corresponding Beckhoff terminal on the terminal box that has the following properties:

- analog input terminal
- 2 channels
- Condition monitoring (IEPE)
- 16-bit resolution
- +\- 5 V

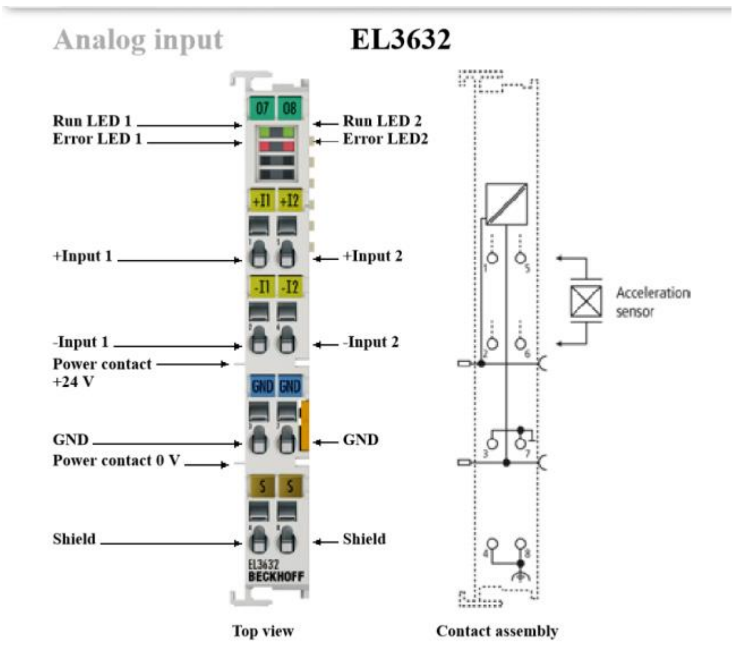


Figure 81: EL 3632

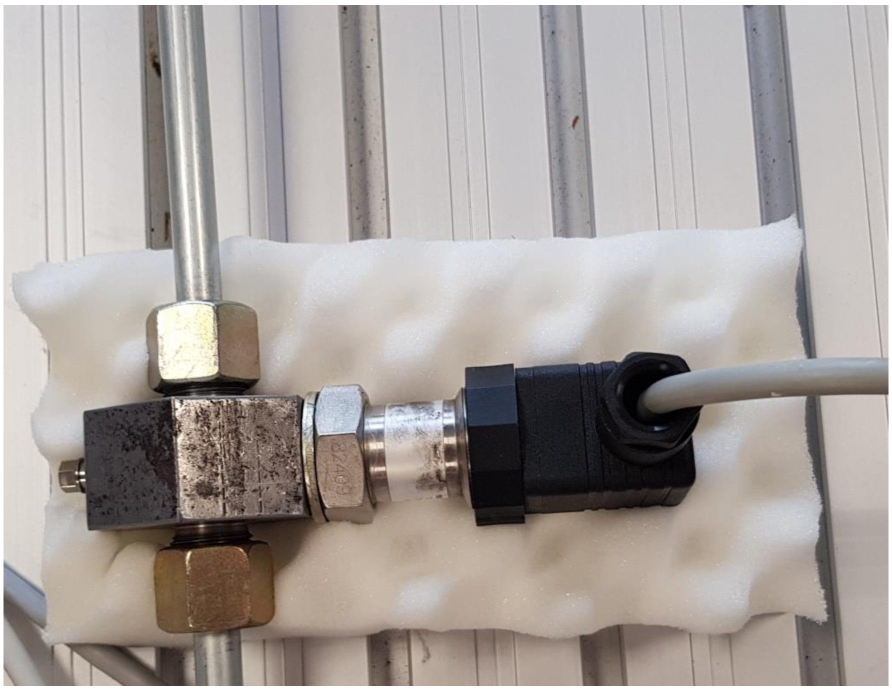


Figure 82: The PCB pressure sensor

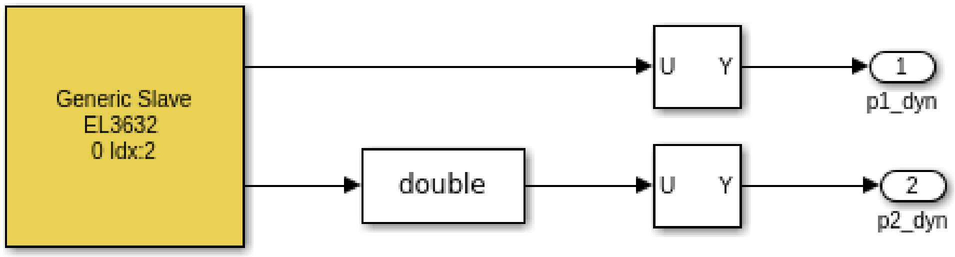


Figure 83: EL 3632 block in Simulink

9.5.4 Static pressure

EL 3702

As mentioned before, in order to guarantee that the target pressure is reached, an electro pneumatic regulator maintains constant output pressure in compressed-air systems regardless of variations in the input pressure or output flow.

After inserting the value of the target pressure, the regulator takes some time to reach it. During this time, the tank's relative pressure is changing and in order to measure its value, a "Keller gauge pressure sensor (25Y series)" (**Figure 84**) is used.

Figure 84: The 25Y series Keller pressure sensor

The dedicated terminal in the Beckhoff terminal box takes this value and passes it to the control program. Whenever the target pressure is reached, the control algorithm starts working.

This block (**Figure 86**) represents the corresponding Beckhoff terminal on the terminal box that has the following properties:

- 2 channels
- analog input
- -/+ 10 V with oversampling
- 16-bit resolution

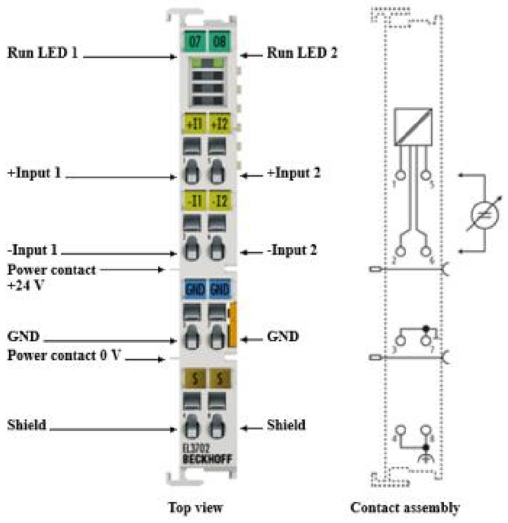


Figure 85: EL 3702 terminal

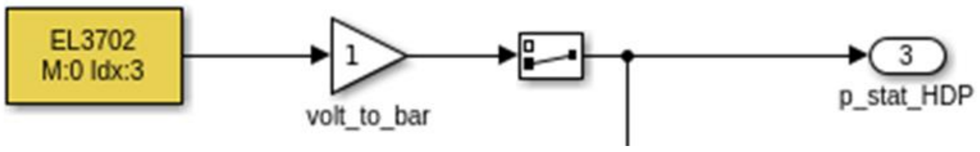


Figure 86: The EL 3702 Simulink block with the proper gain

9.5.5 The incremental Encoder interface

EL 5101

In order to obtain the current rotational speed of the electric motor (in rpm), an incremental rotary encoder interface is used. A rotary encoder, also called a shaft encoder, is an electro-mechanical device that converts the angular position or motion of a shaft or axle to analog or digital output signals. An RS 795-1126 incremental encoder (**Figure 87**) was used on the test-rig to measure the rotational speed and the angle of the motor shaft in case the latter is needed. It has the following properties:

- 512 pulses/revolution
- Incremental
- Maximum revolutions: 6000 rpm
- Resolution: 2500 ppr

Figure 87: The RS 795-1126 incremental rotary encoder interface

The dedicated EtherLAB block that represents the Beckhoff terminal in the terminal box is the EL 5101 block (**Figure 88**); this terminal has the following properties:

- An interface for the direct connection of incremental encoders with differential inputs.
- 16/32-bit switchable counter
- 16/32-bit latch for the zero pulse
- Commands: read, set and enable

- Resolution: 1/256 bit micro increments

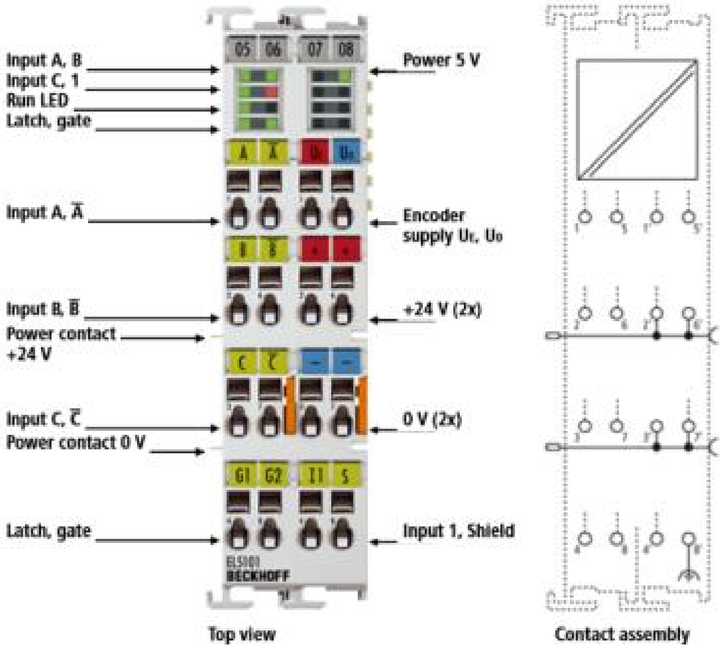


Figure 88: The EL 5101 Beckhoff terminal

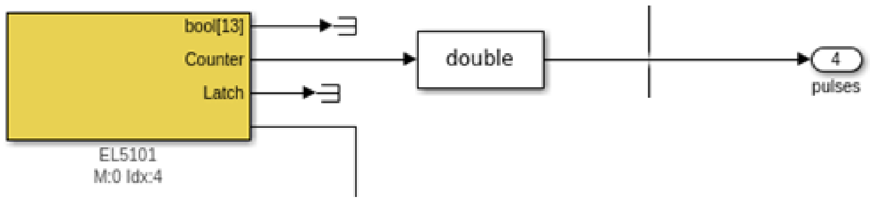


Figure 89: The EL 5101 Simulink block

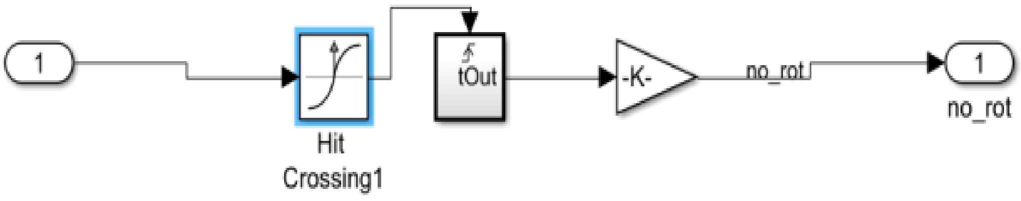


Figure 90: The algorithm to calculate the number of rotations

When the incremental encoder interface generates pulses, this algorithm (**Figure 90**) calculates this number of pulses and their frequency in order to obtain the current rpm value of the electric motor. The gain block here holds a value of 60/512 because the incremental encoder interface generates 512 pulses per revolution and this number is multiplied by 60 to get the number of rotations per minute.

9.5.6 Fluid Temperature

EL 3204

In order to measure the temperature of the fluid, *a Siemens KTY 19-6* M/Z temperature sensor (**Figure 91**) was used.

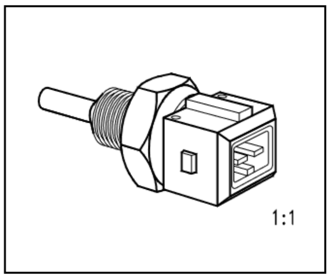


Figure 91: The Siemens temperature sensor

The dedicated EtherLAB block that represents the Beckhoff terminal in the terminal box is the EL 3204 block (**Figure 93**); this terminal has the following properties:

- Analog input
- 4 channels
- PT100 (RTD)
- 0.1 °C in the temperature range of PT100 sensors

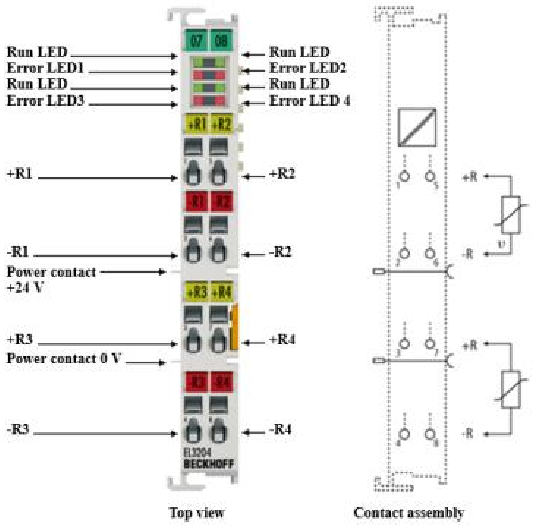


Figure 92: The EL 3204 Beckhoff Terminal

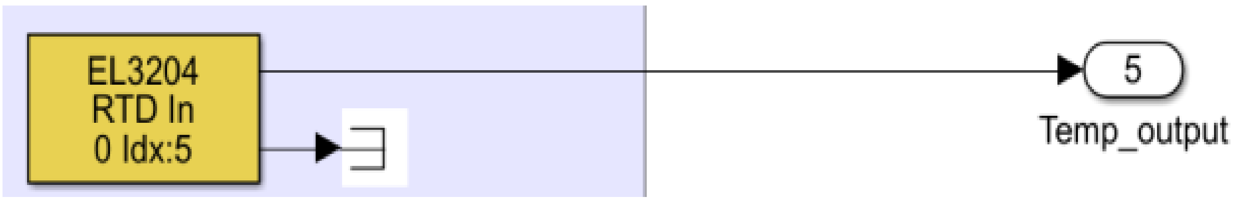


Figure 93: The EL3204 Simulink block

9.5.7 The needle factor and the Faulhaber motor

This is the most important part of the whole controller algorithm as well as the process of driving the needle. The RaLa was equipped with a manual hand wheel (**Figure 99**) which was responsible of guiding the linear motion of the needle, but this hand wheel was replaced with the *Faulhaber motor drive MCS3268G024BX4 ET BS32-2.0* (**Figure 100**).

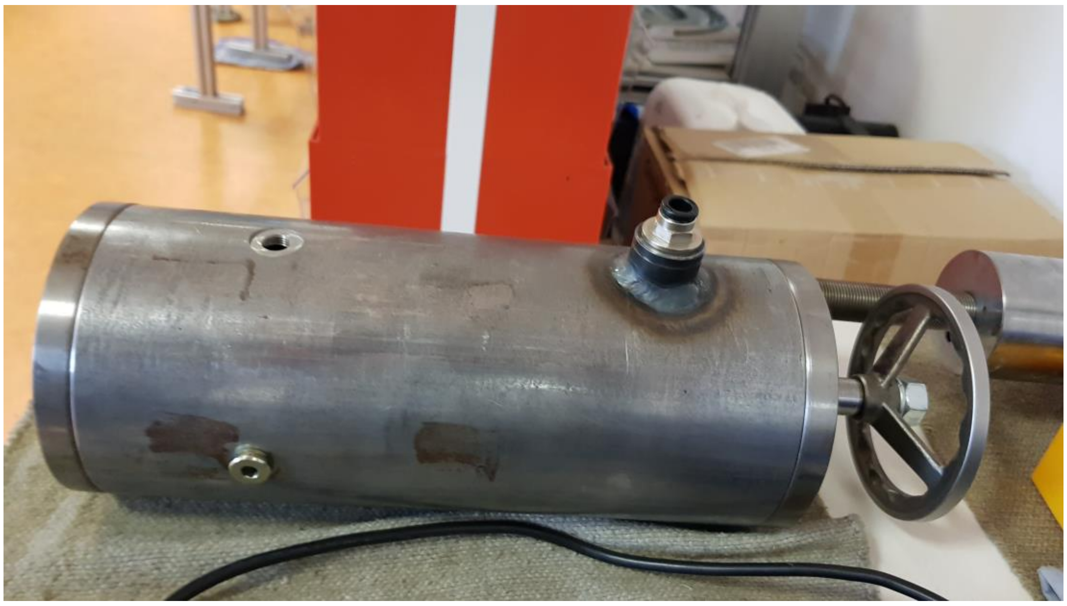


Figure 94: RaLa with hand wheel

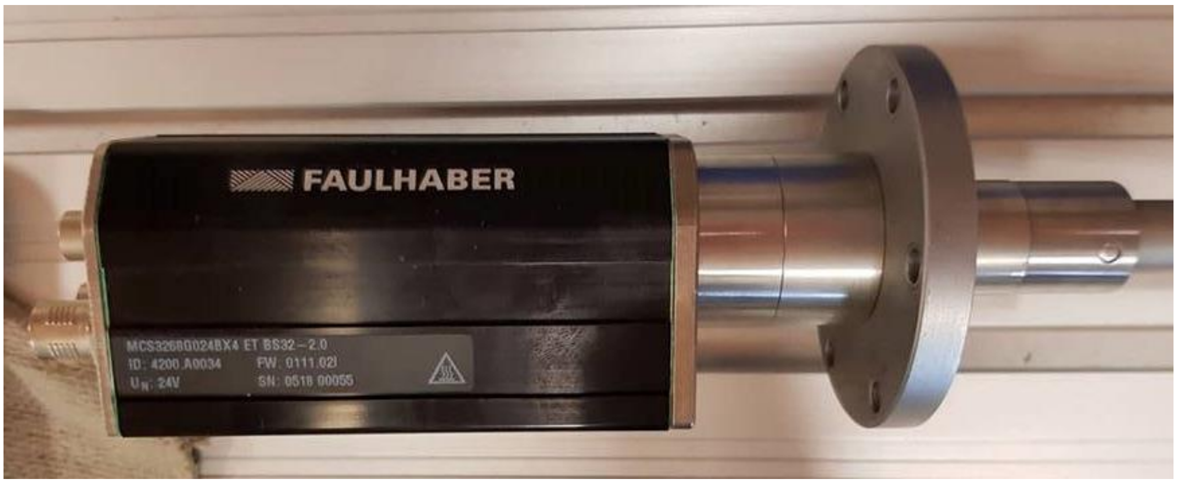


Figure 95: Faulhaber Drive Motor

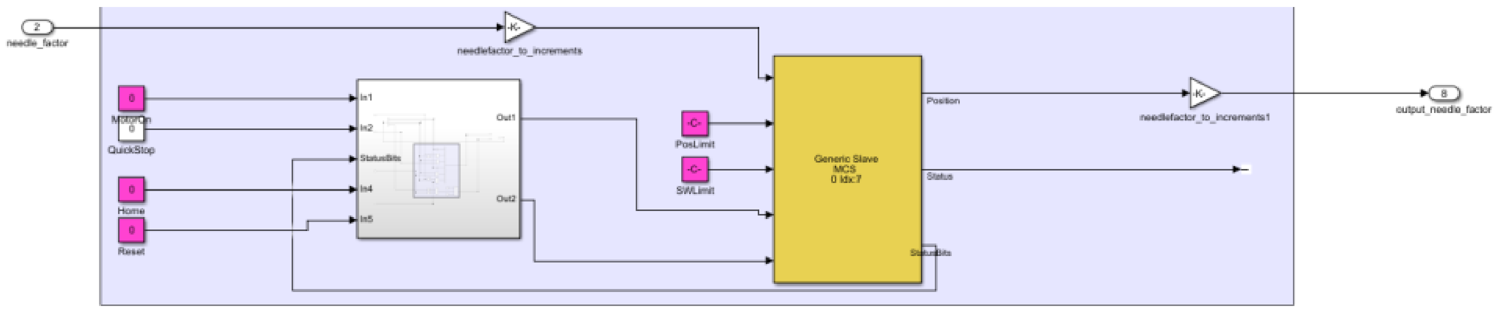
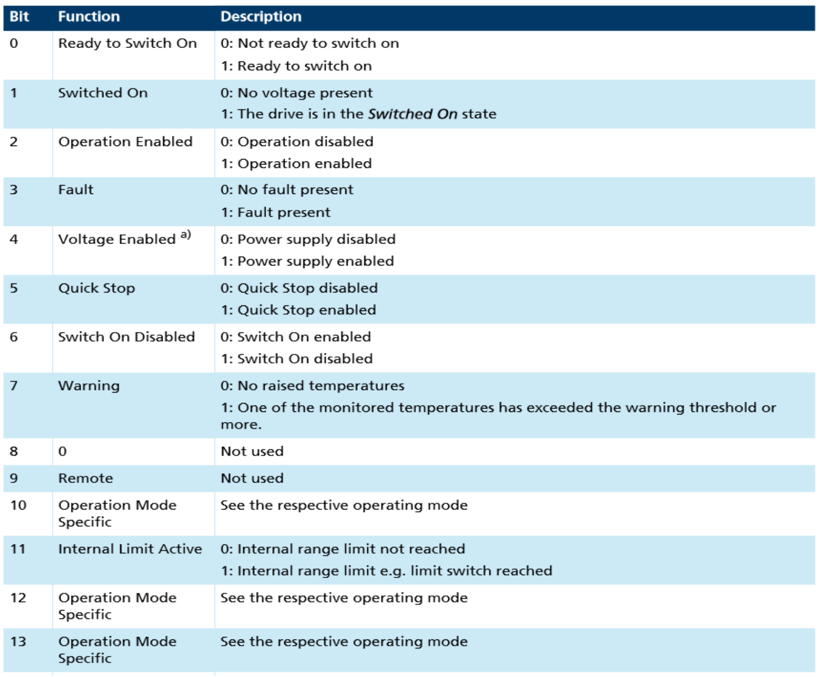


Figure 96: The Faulhaber Drive Motor Simulink model

The MCS (Motion Control System) Simulink block alongside the grey subsystem were ordered from the IGH Gmbh Company that developed the EtherLAB blocks because the dedicated block can't be found in their library. According to the documentation, these are some information about the inputs and outputs that can be seen in **Figure 96**:

- Meaning of the bits in the statusword:



Bit	Function	Description
0	Ready to Switch On	0: Not ready to switch on 1: Ready to switch on
1	Switched On	0: No voltage present 1: The drive is in the *Switched On* state
2	Operation Enabled	0: Operation disabled 1: Operation enabled
3	Fault	0: No fault present 1: Fault present
4	Voltage Enabled [a)]	0: Power supply disabled 1: Power supply enabled
5	Quick Stop	0: Quick Stop disabled 1: Quick Stop enabled
6	Switch On Disabled	0: Switch On enabled 1: Switch On disabled
7	Warning	0: No raised temperatures 1: One of the monitored temperatures has exceeded the warning threshold or more.
8	0	Not used
9	Remote	Not used
10	Operation Mode Specific	See the respective operating mode
11	Internal Limit Active	0: Internal range limit not reached 1: Internal range limit e.g. limit switch reached
12	Operation Mode Specific	See the respective operating mode
13	Operation Mode Specific	See the respective operating mode

Figure 97: Meaning of the bits in the statusword

- Position Limit: from 0 to 32000 increments.
- Software Limit: from 0 to 10 or 20 increments less than the position limit.
- The controlword controls the transitions while the statusword shows the states.

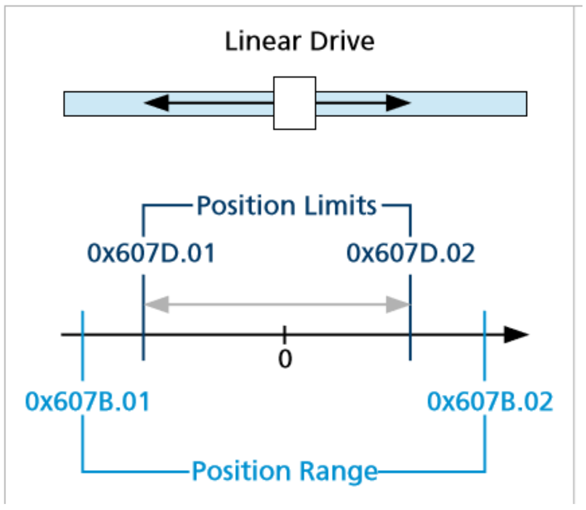


Figure 98: Position Limits and Range

Internally the position is calculated in increments directly in the resolution of the position sensor used. The total number of increments from the lower end stop to the upper one is 32000 for a length of 10 mm of travel at the clam nut, which results in 3200 increments/mm of stroke. The number of revolutions of the spindle doesn't matter in this case as the controller algorithm moves the needle according to its position as a result of the data it receives, so it merely depends on the number of increments.

As you can see in **Figure 96**, the gain block *needlefactor_to_increments* holds a value of :

$\frac{maximum\ number\ of\ increments}{maximum\ value\ of\ needle\ factor} = \frac{32000}{2}$ but the maximum number of increments isn't always equal to 32000 increments which is why a calibration of the motor is required before starting any new measurement.

The Faulhaber Motor Drive calibration

The calibration in **Figure 99** was done to know the number of increments in the minimum and maximum stroke because the length of the stroke wasn't known. In general, the calibration has to be done at every measurement to know exactly the minum and maximum number of increments in order to insert them as parameters in the controller algorithm to ensure an exact and precise output as an extra micrometer can affect our results.

In addition to that, when the motor is turned on and off, a minor change in the position of the eedle, either done deliberately or no, can change the minimum and maximum number of increments inside the motor.

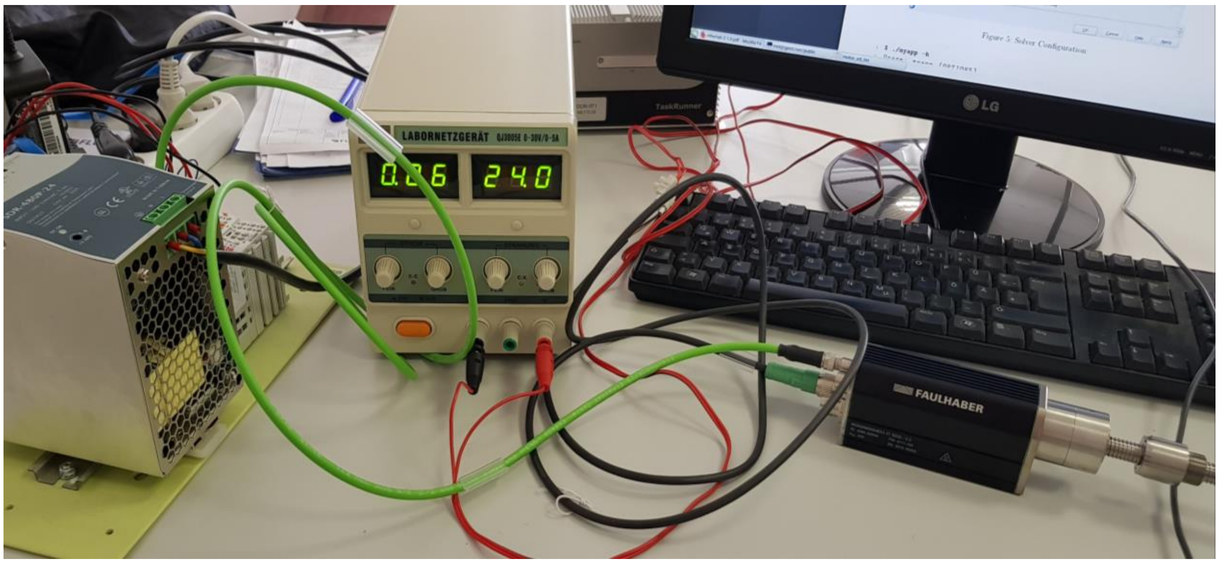
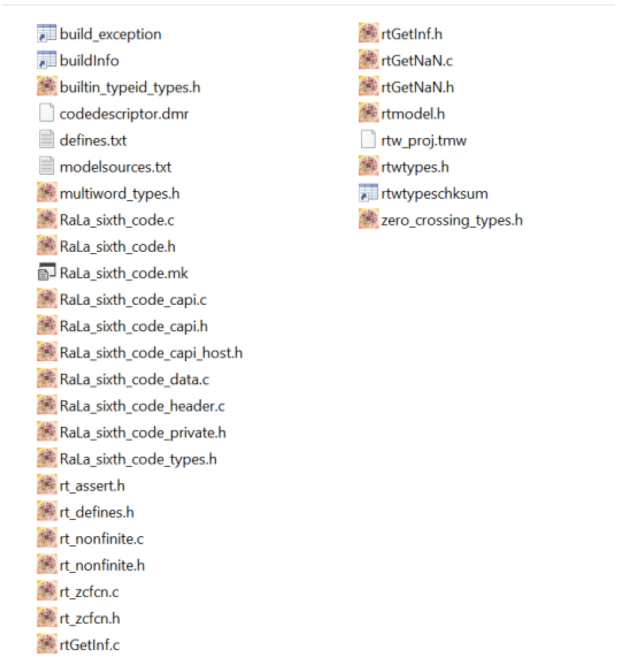


Figure 99: The Faulhaber motor calibration setup

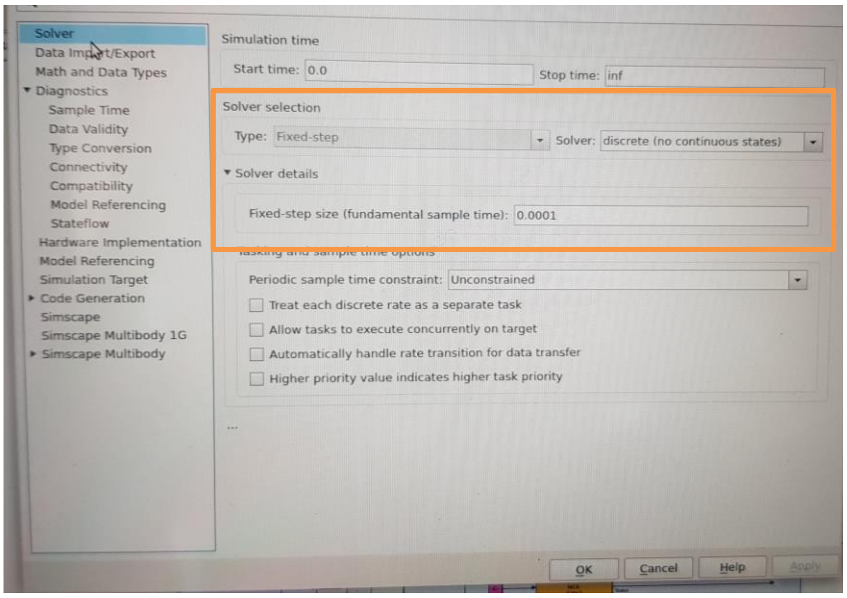
Eventually, the Simulink model is built and the generated code contains the necessary source and header files in order to be implemented and executed on a Linux OS. The folder containing these files can be found under the name "<Model name>_etl_hrt" and can be found in the same folder where the Simulink model exists. **(Figure 100)**



```
build_exception                  rtGetInf.h
buildInfo                        rtGetNaN.c
builtin_typeid_types.h           rtGetNaN.h
codedescriptor.dmr               rtmodel.h
defines.txt                      rtw_proj.tmw
modelsources.txt                 rtwtypes.h
multiword_types.h                rtwtypeschksum
RaLa_sixth_code.c                zero_crossing_types.h
RaLa_sixth_code.h
RaLa_sixth_code.mk
RaLa_sixth_code_capi.c
RaLa_sixth_code_capi.h
RaLa_sixth_code_capi_host.h
RaLa_sixth_code_data.c
RaLa_sixth_code_header.c
RaLa_sixth_code_private.h
RaLa_sixth_code_types.h
rt_assert.h
rt_defines.h
rt_nonfinite.c
rt_nonfinite.h
rt_zcfcn.c
rt_zcfcn.h
rtGetInf.c
```

Figure 100: The generated source and header files

Before building the model, some properties have to be changed by clicking on the toolbar: Simulation > Model Configuration Parameters then a window is going to show up which contains the parameters and properties that can be adjusted. In the following figures, the fields that were modified were encapsulated with a box.



Simulation time

Start time: 0.0 Stop time: inf

Solver selection

Type: Fixed-step Solver: discrete (no continuous states)

Solver details

Fixed-step size (fundamental sample time): 0.0001

Tasking and sample time options

Periodic sample time constraint: Unconstrained

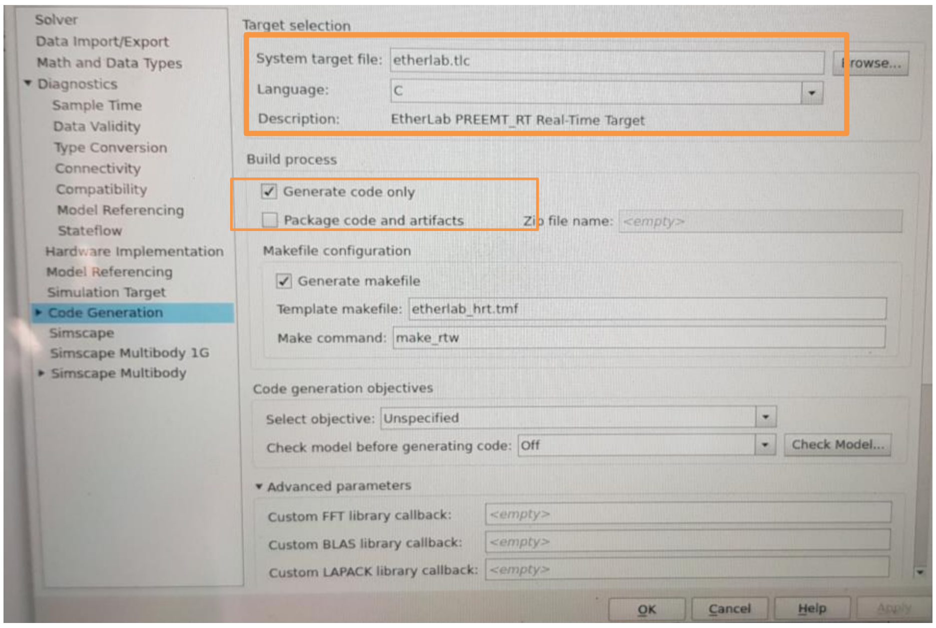
☐ Treat each discrete rate as a separate task

☐ Allow tasks to execute concurrently on target

☐ Automatically handle rate transition for data transfer

☐ Higher priority value indicates higher task priority

Figure 101: Solver parameters



Target selection

System target file: etherlab.tlc

Language: C

Description: EtherLab PREEMPT_RT Real-Time Target

Build process

☑ Generate code only

☐ Package code and artifacts Zip file name: <empty>

Makefile configuration

☑ Generate makefile

Template makefile: etherlab_hrt.tmf

Make command: make_rtw

Code generation objectives

Select objective: Unspecified

Check model before generating code: Off

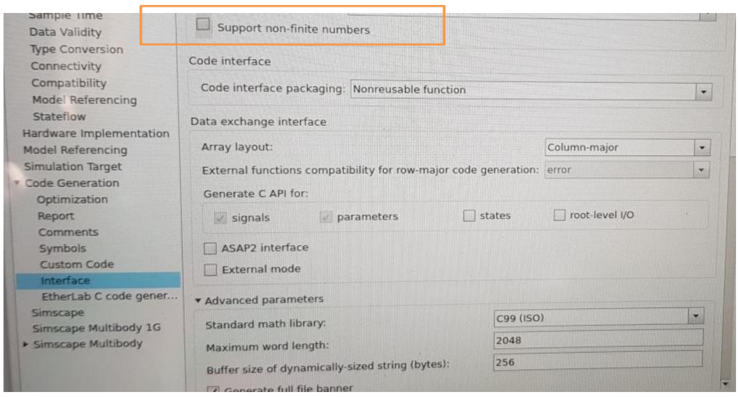
Advanced parameters

Custom FFT library callback: <empty>

Custom BLAS library callback: <empty>

Custom LAPACK library callback: <empty>

Figure 102: Code Generation parameters



☐ Support non-finite numbers

Code interface

Code interface packaging: Nonreusable function

Data exchange interface

Array layout: Column-major

External functions compatibility for row-major code generation: error

Generate C API for: ☑ signals ☑ parameters ☐ states ☐ root-level I/O

☐ ASAP2 interface

☐ External mode

Advanced parameters

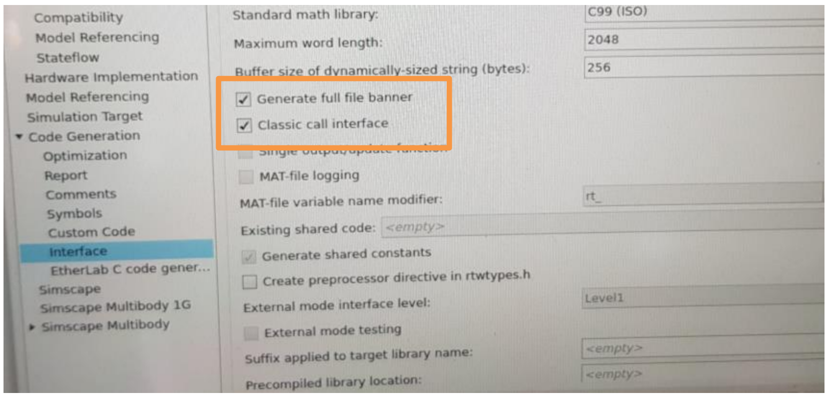
Standard math library: C99 (ISO)

Maximum word length: 2048

Buffer size of dynamically-sized string (bytes): 256

☑ Generate full file banner

Figure 103: Interface Parameters



Standard math library: C99 (ISO)
Maximum word length: 2048
Buffer size of dynamically-sized string (bytes): 256
☑ Generate full file banner
☑ Classic call interface
☐ Single output/update function
☐ MAT-file logging
MAT-file variable name modifier: rt_
Existing shared code: <empty>
☑ Generate shared constants
☐ Create preprocessor directive in rtwtypes.h
External mode interface level: Level1
☐ External mode testing
Suffix applied to target library name: <empty>
Precompiled library location: <empty>

Figure 104: Interface Parameters (Additional Parameters)

The “Classic call interface” must be checked while the “Support non-finite numbers” has to be left unchecked or else a “call to rt_nonfinite.h” error occurs.

10 Conclusion

The thesis discussed the development and test of an automated RaLa needle positioning controller algorithm in order to adapt the device’s characteristic impedance to the impedance of the pipeline so that a reflection-free state in the line can be achieved in order to obtain accurate results from the test-rig measurements done on the pump. It included an analysis of the hybrid pump model used to incorporate the dynamic behavior of a hydraulic displacement pump into the time based simulation. The thesis also discussed, in detail, the qualitative methodological approach for the development of the hydraulic model as well as the controller algorithm used to achieve a full automation of the needle position depending on the user’s preferences. The following conclusions can be drawn:

- The simulations done using DSHplus and Simulink have demonstrated the robustness of the algorithm and its ability to deal with erroneous input parameters inserted by the user.
- The algorithm was tested on different kinds of pumps on all the functional ranges of the system pressure and the rotational speed of the motor which are used in the company during the tests.
- The reflection-free state was achieved after driving the needle to the proper position automatically.
- Further tests can be done on the real test-rig and modifications can be applied on the algorithm accordingly depending on the results of these tests.

11 References

(1) Title: " Investigation on Cardiovascular Risk Prediction Using Physiological Parameters" Authors: Wan-Hua Lin, Heye Zhang and Yuan-Ting Zhang

(2) Wright (1983) - Matched Impedance to Control Fluid Transients

(3) Hybrid Pump Model for 1D Hydraulic System Simulation : HeikoBaum*, KlausBecker** and Axel Faßbender** (FLUIDONGesellschaftfürFluidtechnikmbH, Jülicher-Straße338a, 52070 Aachen, Germany*, Cologne University of Applied Sciences, Faculty of Automotive Systems and Production Engineering, BetzdorferStraße2, 50679 Köln, Germany*)

(4) https://www.fluidon.com/en/dshplus

(5) Title:" **Betriebsanleitung**" (p:2) Author: **Dirk de Ben**

(6) http://m.beckhoff.com/english/beckhoff/company.html

(7) https://en.wikipedia.org/wiki/EtherCAT

(8) https://etherlab.org/en/what.php