

ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

---

SCHOOL OF ENGINEERING AND ARCHITECTURE

*DEPARTMENT OF ELECTRICAL, ELECTRONIC AND INFORMATION  
ENGINEERING "GUGLIELMO MARCONI" (DEI)*

MASTER DEGREE IN AUTOMATION ENGINEERING  
8891

GRADUATION THESIS

in

MECHATRONICS SYSTEMS MODELING AND CONTROL M

# Motion control development in the TIA Portal for the MS260 binding and wrapping machine

CANDIDATE:  
Andrea Balboni

SUPERVISOR:  
*Eminent prof.* Alessandro Macchelli

CO-SUPERVISOR:  
*eng.* Michele Padovani

Academic year  
2019/2020

Session  
III



# Ringraziamenti

Vorrei ringraziare, innanzitutto, il mio supervisore Michele Padovani per il supporto e per avermi guidato durante lo sviluppo di questo progetto. Una particolare menzione anche per il mio relatore Alessandro Macchelli per avermi offerto questa opportunità e per la sua sempre presente disponibilità.

Vorrei ringraziare, inoltre, tutti coloro con cui ho potuto condividere questi anni universitari e che da colleghi sono diventati amici di vita a tutti gli effetti; da coloro con cui ho condiviso l'intero percorso universitario, Luca, Giova, Ros, Lory, Martina, Euseb, chi ho conosciuto alla magistrale, Baruz, e chi con cui ho trascorso solamente la triennale, Mula, Steve, Alle Alby e Shou.

Non dimentico, inoltre, chi ho avuto l'opportunità di conoscere durante questi stessi anni al di fuori dell'università, Nick, Werther e Samu.

Vorrei ringraziare anche, troppi da poter menzionare tutti, coloro con cui ho trascorso gli anni delle scuole primarie e gli anni da calciatore.

Ringrazio la mia ragazza Federica e la sua famiglia che, nonostante siano entrati da poco nella mia vita, rappresentano una parte importante di essa e con cui ho potuto condividere alcuni dei momenti più importanti del mio percorso.

Infine, ma primi tra tutti, vorrei ringraziare in particolar modo la mia famiglia, i miei genitori Carlo e Nadia e mia sorella Alice, per avermi sempre sostenuto e per avermi dato la splendida opportunità di completare la mia formazione. Ringrazio anche i miei zii, Luca e Cristina, mio cugino Matteo e mia nonna Alba con cui ho avuto la possibilità di crescere.

# Abstract

La MS260 è una macchina legatrice e avvolgitrice. È una macchina fine-linea studiata e creata dalla IMA S.p.A e in particolare dalla divisione dedicata ai fine linea, cioè IMA BFB.

Il lavoro di tesi è concentrato sullo sviluppo della macchina utilizzando esclusivamente hardware e software Siemens. In particolare la MS260 presa in considerazione è suddivisa in due parti. La parte relativa al PLC è caratterizzata da hardware Siemens con relativo algoritmo sviluppato tramite software TIA Portal. Parallelemente la parte di motion control è interamente caratterizzata da elementi legati al mondo Schneider. Drive e motori appartengono alla suddetta azienda, così come l'algoritmo è sviluppato tramite il software nativo MachineExpert.

L'obiettivo della tesi è, dunque, suddiviso in due parti. In primis, è necessario studiare le caratteristiche della macchina. In questo modo è possibile poter selezionare un hardware adeguato per il nuovo codice. Lo studio delle performance dei motori consente un scelta ottimale degli azionamenti dal catalogo Siemens. Inoltre la loro sostituzione richiede anche il cambiamento del PLC, passando a un componente capace di gestire un controllo degli assi più avanzato. In secundis, l'algoritmo di motion control deve essere sviluppato e integrato. Per farlo vengono sfruttate alcune librerie di Siemens mentre altre devono essere sviluppate per poter realizzare funzioni non native. L'integrazione finale comporta l'adattamento del codice PLC alla nuova organizzazione dell'algoritmo di motion e la modifica parziale del software HMI per l'aggiunta delle nuove funzionalità.

# Abstract

The MS260 is a biding and wrapping machine. It is an end-of-line machine designed and created by IMA S.p.A and, in particular, by the division dedicated to the end-of-line, i.e. IMA BFB.

The thesis work is concentrated on the development of the machine by exclusively using Siemens hardware and software. In particular, the MS260 analysed is divided into two parts. The PLC part is characterized by Siemens hardware with the relative algorithm developed through the TIA Portal software. At the same time, the motion control part is entirely characterized by elements linked to the Schneider world. Drives and motors belong to the aforementioned company, as well as the algorithm is developed through the native software MachineExpert.

The objective of the thesis is, then, divided into two parts. First of all, it is necessary to study the characteristics of the machine. This allows to select a suitable hardware for the new code. The study of the motor performances allow an optimal choice of the actuators on the Siemens catalogue. Furthermore, the replacement of the drives requires also the change of the PLC, passing to an hardware capable of handling more advanced control commands. Secondly, the motion control algorithm must be developed and integrated. To do this, some Siemens libraries are exploited while others must be created in order to exploit non-native functions. The final integration foresees the adaptation of the PLC code to the new organization of the motion algorithm and the partial modification of the HMI software to add the new functionalities.

# Contents

<b>Introduction</b>	<b>10</b>
<b>1 Software architecture</b>	<b>12</b>
1.1 Machine operative conditions . . . . .	13
1.2 PLC Siemens structure . . . . .	14
1.3 Structure of the Schneider motion control . . . . .	17
1.3.1 Machine module organization . . . . .	19
1.3.2 Sub-modules organization . . . . .	20
1.4 Schneider alarm managing . . . . .	24
1.5 PLC-Drive connection . . . . .	30
<b>2 Hardware structure</b>	<b>34</b>
2.1 Schneider catalogue . . . . .	34
2.2 MS260 motors description . . . . .	36
2.3 Siemens hardware . . . . .	38
2.4 Siemens PLC . . . . .	44
2.5 Siemens motor selection . . . . .	45
<b>3 Structure of the TIA Portal project</b>	<b>67</b>
3.1 Code structure . . . . .	67
3.2 Technology objects . . . . .	70
3.2.1 TO_Axes . . . . .	71
3.2.2 TO_OutputCam . . . . .	73
3.2.3 TO_Cam . . . . .	75
3.2.4 Organization blocks . . . . .	75
3.3 Drive unit . . . . .	78
3.3.1 Hardware definition . . . . .	78
3.3.2 Parameters access . . . . .	83
3.4 Libraries . . . . .	85
3.4.1 IsAxisReady . . . . .	85
3.4.2 LAxisCtrl . . . . .	87
3.4.3 LCamHdl . . . . .	90
3.4.4 Crank module . . . . .	91
<b>4 Motion Control algorithm</b>	<b>96</b>
4.1 Implementation of the motion control . . . . .	96
4.2 Motion structure . . . . .	97

4.2.1	MM_Machine . . . . .	98
4.2.2	Sub-modules organization . . . . .	99
4.3	Axis module . . . . .	101
4.4	Initialization and blocks execution . . . . .	105
4.5	Sub-module bridges . . . . .	107
4.6	Motion control actions . . . . .	108
4.6.1	Motion block structure and management . . . . .	108
4.6.2	SM_Layer . . . . .	112
4.6.3	SM_Stretch . . . . .	117
4.7	Format change management . . . . .	121
4.7.1	Layer sub-module format change . . . . .	121
4.7.2	Stretch sub-module format change . . . . .	124
4.8	Motion control logic . . . . .	129
4.9	Errors . . . . .	132
4.9.1	Error management FB . . . . .	132
4.9.2	Error logic . . . . .	135
4.10	HMI panel . . . . .	139
4.11	Simulation . . . . .	143
<b>Conclusions</b>		<b>146</b>
<b>A Crank module code</b>		<b>148</b>
<b>B HMI libraries</b>		<b>153</b>
<b>Bibliography</b>		<b>162</b>

# List of Figures

1.1	PLC Siemens Tree. . . . .	14
1.2	Modules Organization. . . . .	14
1.3	Modules Programs. . . . .	16
1.4	Schneider Tree. . . . .	17
1.5	Schneider program structure. . . . .	18
1.6	POUs Tree. . . . .	18
1.7	POUs SFC Tree. . . . .	20
1.8	Error Handler acquisition. . . . .	25
1.9	Error Handler transmission. . . . .	25
1.10	Reaction Table. . . . .	25
1.11	Reaction Table layer structure. . . . .	26
1.12	Reaction Table: layer composition. . . . .	26
1.13	Reaction Table: layer meaning. . . . .	27
1.14	Reaction Table: layer modification. . . . .	27
1.15	Error definition example. . . . .	28
1.16	Error handling example. . . . .	28
1.17	Example of exceptions transmission. . . . .	29
1.18	OM structure. . . . .	30
1.19	Line Level structure. . . . .	31
1.20	Data Transmission Schneider. . . . .	31
1.21	Data Transmission PLC. . . . .	32
1.22	Vitality-Check of PLC. . . . .	33
2.1	SH3 servo-motors nomenclature. . . . .	36
2.2	DT Configurator Initialization. . . . .	40
2.3	DT Configurator Parameters. . . . .	41
2.4	DT Configurator Alternatives. . . . .	41
2.5	SIZER Initialization. . . . .	42
2.6	SIZER parametrization. . . . .	43
2.7	SIZER Motor Characteristics Curves. . . . .	43
2.8	ET 200 SP Open Controller. . . . .	44
2.9	Advanced controllers. . . . .	45
2.10	Master of the layer axis at 55% speed in no load run. . . . .	48
2.11	Layer axis at 55% speed in no load run. . . . .	48
2.12	Master of the machine axes at 24 u/min in no load run. . . . .	48
2.13	Pusher axis 24 u/min in no load run. . . . .	49
2.14	Film Pull axis 24 u/min in no load run. . . . .	49



---

2.15	Crank axis 24 u/min in no load run. . . . .	49
2.16	Virtual Crank axis at 24 u/min in no load run. . . . .	50
2.17	Master of the layer axis at 75% speed in no load run. . . . .	50
2.18	Layer axis 60 u/min in no load run. . . . .	50
2.19	Master of the machine axes at 60 u/min in no load run. . . . .	51
2.20	Pusher axis 60 u/min in no load run. . . . .	51
2.21	Film Pull axis 60 u/min in no load run. . . . .	51
2.22	Crank axis 60 u/min in no load run. . . . .	52
2.23	Virtual Crank axis at 60 u/min in no load run. . . . .	52
2.24	Data sheet of 3070 Schneider actuator. . . . .	53
2.25	Data sheet of 3100 Schneider actuator. . . . .	54
2.26	Layer Operative Points. . . . .	56
2.27	Layer 1FK7040-2AK7. . . . .	57
2.28	Layer 1FK7034-2AK7. . . . .	57
2.29	Layer 1FK2204-5AF0. . . . .	58
2.30	Pusher Operative Points. . . . .	59
2.31	Pusher 1FK7034-5AK7. . . . .	60
2.32	Pusher 1FK2204-6AF0. . . . .	60
2.33	Crank Operative Points. . . . .	62
2.34	Crank 1FK7061-4CH7. . . . .	62
2.35	Crank 1FK7060-2AH7. . . . .	63
2.36	Crank 1FK7060-2AF7. . . . .	63
2.37	Crank 1FK7064-4CF7. . . . .	64
2.38	Film Pull operative Points. . . . .	65
2.39	1FK7015-5AK7. . . . .	66
2.40	1FK7022-5AK7. . . . .	66
3.1	Project structure. . . . .	69
3.2	TO_OutputCam Distance-based when start position is lower than the end position. . . . .	74
3.3	TO_OutputCam Distance-based when start position is greater than the end position. . . . .	74
3.4	TO_OutputCam Time-based. . . . .	74
3.5	OB cycle priorities. . . . .	77
3.6	Topology view. . . . .	78
3.7	Network view. . . . .	79
3.8	Device view of the PLC. . . . .	80
3.9	Device view of the drive. . . . .	81
3.10	Modules Telegram. . . . .	82
3.11	TO_Axes Sensors. . . . .	86
3.12	Example of Enable activation for MS_Power. . . . .	87
3.13	Example of LAxisCtrl FB. . . . .	89
3.14	Crank module structure. . . . .	93
4.1	Siemens motion control layers. . . . .	98
4.2	Sub-module Structure. . . . .	99
4.3	Drive error management. . . . .	104

4.4	Axis Parameter. . . . .	104
4.5	Drive error management. . . . .	105
4.6	FB organization of the motion actions. . . . .	110
4.7	Bypass Solution. . . . .	115
4.8	Format Change Organization. . . . .	122
4.9	Buffer geometry. . . . .	123
4.10	Layer axis cams. . . . .	124
4.11	Pusher axis cam. . . . .	126
4.12	Crank axis cam. . . . .	127
4.13	Virtual Crank axis cam. . . . .	128
4.14	Film pull axis cam. . . . .	128
4.15	Interpolation code of the alternative cam. . . . .	132
4.16	Fault activation examples. . . . .	138
4.17	Show cam page. . . . .	140
4.18	PLC FB for DB import-export. . . . .	142
4.19	HMI DB import-export page. . . . .	143
4.20	PLC performances. . . . .	145

# Introduction

## Motivations

This thesis project deals with the possibility of creating a complete Siemens machine. Indeed, like many other machines created by IMA BFB, the MS260 analysed is characterized by a PLC control part built from Siemens software and components and an actuation part realized with Schneider software and hardware. Therefore, it was considered to convert the latter part by exploiting the Siemens software and hardware.

## State of the art

Nowadays, many software are available for the creation of PLC algorithms. Especially in IMA BFB, three are used for the development of their end-of-line machines: TIA Portal by Siemens, MachineExpert by Schneider and Rockwell. In addition to them, many others are available, like the B&R software also used in other divisions of IMA. Each of them is characterized by their organization of the algorithm and languages. Moreover, each has functionalities not supported or not implemented by the other software, like in this thesis is going to be described for the two of them considered. Therefore, there is not a better choice. In the IMA BFB division, the more exploited platforms are Siemens and Schneider. They are coexistent in many machines developed, in particular in the smaller ones. As the MS260 is one of them, it is composed of both, making the development of a complete Siemens machine a new approach.

## Proposed solution

For the realization of such a goal, some steps are followed. An initial study of the machine was first performed. The components present and the operations they perform must be analysed to gain a better understanding of the development possibilities. This represents, indeed, the first occupation of my internship at IMA BFB. First, I studied the MS260 in its entirety. This is what the opening chapters of this thesis describe. In the first one, the software part is analysed. There, the two algorithms that the PLC and the drive execute are presented. It is an important initial step to understand what and how it was developed afterward. Then, in the second chapter, I enter into the detail of the hardware part. This implies the description of the actuators and the PLC mounted on the old version of the machine. This allows explaining the tests performed to find out the performances of the motors. Their study makes it

---

possible to see the limits to select the proper alternatives. Indeed, the Siemens actuators selection follows from them. I also introduced a little digression on the new PLC choice, as it is forced to be more performant in order to execute the new advanced motion control commands.

The successive step is dedicated to the study in depth of the TIA Portal software. To take full advantage of its possibilities, you need to understand how it manages the elements of the software and what it allows and does not allow doing. This is what the third chapter tries to explain. There, in parallel, I describe also the organization of the new motion control algorithm. Therefore, it contains the integrated and custom libraries used.

This leads to the last chapter where the code realised is presented. It is described step by step in all its functions by recalling the elements analysed previously.

# Chapter 1

## Software architecture

The MS260 is a binding and wrapping machine, that is, it picks up the individual packs from the upper machine and collects them in bundles of predetermined dimensions through a wrapping film.

To understand what was done in the motion control code developed for the project, an analysis of the machine already built is first performed. In particular, this chapter presents the operations of the machine to give an idea of the work it does. Subsequently, the software control architecture of the machine is analysed which is divided into two main parts:

- the first is represented by the Siemens PLC code. This is aimed at checking the status and at managing machine alarms;
- the second contains the motion control executed by the Schneider drive. Accordingly, this part is aimed at the control of the motors' behaviour.

In parallel to these two, the HMI software is also present. It allows the operator to access the machine status and behaviours from an external panel. As it is for the first of the two parts mentioned above, the main structure is not modified too much because it is not the main topic of the project. But, similarly, how some changes are made to support the libraries built during project development are also analysed.

In the last section of this chapter, the connection between the status and the motion control parts is also described. This is very important as it represents one of the main topics in terms of comparisons between the Schneider and the entire Siemens project.

---

## 1.1 Machine operative conditions

The MS260 is a wrapping and binding machine. Consequently, its main task is to take the individual packages coming from an upper machine through the in-feed belt and to generate bundles of specified size and composition by wrapping them with a film. The specifications of the bundle can be changed accordingly to the selected format in the HMI panel. In particular, the number of layers and the number of packages per layer are specified. To better understand the functionality of the machine, the hardware components and how they relate to each other will be discussed.

The MS260 analysed can be decomposed in two sub-modules. The first is called SM\_Layering and the second is the SM\_Stretch. The former, as the name suggests, has the job to create the layers of the bundle. To speed up the machine operation, this is performed while the second part of the machine is running. Therefore, a buffer is necessary between them. Within this buffer, the layers are collected as long as their size allows. Before the total height of the collected packages exceeds that of the buffer while the Pusher has not yet finished its operations, the Layer is stopped until the axes of the second part reach a waiting position. Thus, the layers are added to the buffer until the bundle is completed. Only in this case, in fact, the layers are pushed together out of the buffer. In the second part, the bundle is wrapped by the film. To do so, three actuators and two pneumatic pistons are used. Starting from the actuators, the first one is called Pusher. Indeed, its job is to push the collected layers ready to be wrapped towards the sealing motors. One of these is the Crank axis. It is used to seal and cut the film around the bundle. The Pusher moves the bundle all the way to the exit position of the machine. This coincides with a position beyond where the film is stretched. As a result, when the Pusher performs its movement, the bundle has three out of four sides covered by the film. At this point, the two pneumatic pistons are activated. One blocks the bundle in its exit position while the second blocks one side of the film which is stretched between two reels. Therefore, this last actuator blocks the part coming from the upper reel. At this point, the Crank starts its motion to cut and seal the film. During this motion, also the last axis performs its job. This one is called Film Pull. Indeed, in order to create a solid bundle, the film must be pulled. This is the reason why the two pneumatic actuators were activated before the motion of this axis. If not, this actuator, which pulls the film of the lower reel, would also pull the bundle back from its correct position under the Crank operation area. At the end of the process, the Crank has sealed and cut the film, while the Film Pull is released and the pneumatic motors are deactivated. The generated bundle is pushed to the exit by the successive one as in this machine an exit belt control is not present.

## 1.2 PLC Siemens structure

As mentioned above, the PLC software part is organized in order to control and manage the status of the machine.

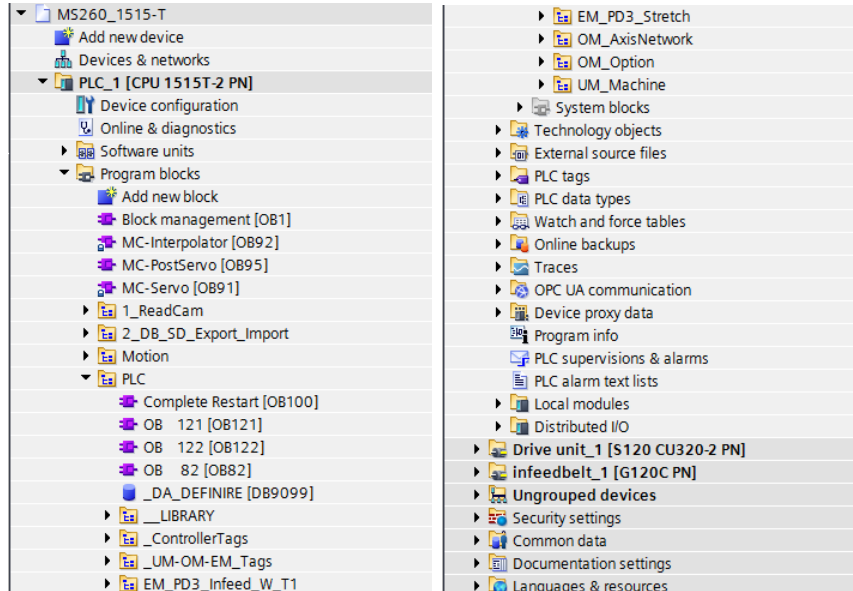


Figure 1.1: PLC Siemens Tree.

The Fig. 1.1 shows how the software is organized inside the *PLC* folder of the *ProgramBlocks* section. The folders of interest are: *EM\_PD3\_Infeed*, *EM\_PD3\_Stretch*, *OM\_AxisNetwork*, *OM\_Option* and *UM\_Machine*. As the names imply, there are different types of modules which allow a structural composition of the PLC. These modules can be divided in three different categories: Overall Modules (OM), Unit Modules (UM) and Equipment Modules (EM). They are organized as Fig. 1.2 shows.

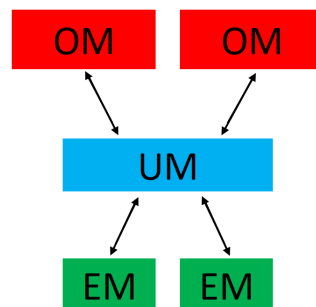


Figure 1.2: Modules Organization.

The Overall Modules represent the upper level of the machine control. Here, the information are managed in order to give to the lower levels of the software reliable and structured data. The first one is the *OM\_AxisNetwork*. It is going to be analysed after the presentation of the Schneider code. It, indeed, contains the PLC-side code for the connection with the drive hardware. Then,

---

for simplicity it is explained along with the Schneider counterpart. The second one is, instead, the *OM\_Option*. This section contains the management of additional operation performed by the PLC which do not affect the execution of the machine. An example is the *AuditTrail* program which is used in order to store the alarms occurred and the related information. Another example is the *Teleservice*.

The Unit Modules are the status control of the machine. In the PLC there is only one UM. It contains most of the programs that the PLC processes. Here, all the functionalities of the machine are analysed; from the warnings' management to the the computation of the machine speed; from the read of the physical inputs to the execution of the of format change initialization requested by the HMI. This module contains all the required algorithms for the control of the machine.

Below this part, the Equipment Modules are present. They are the software programs that manage the action on the motors and act as extensions of the main UM. In particular, in this case two different EMs are present. The *EM\_PD3\_Stretch* controls the status sensors linked to the StretchBanding module of the Schneider motion control. In particular, there are different programs (Fig. 1.3 on the following page). *Cams* is the piece of code which analyses the positions of the sealing section axes. This is useful in order to allow some controls such as the check of the films. The *CrossSealerTemperature* generates, as the name suggests, the reference temperatures and acts as a bridge between the sensors and the HMI. Then, the *FilmBokenCheck* and *FilmEndedCheck* allow the control, the exclusion and the signaling of events related to the film. Indeed, the status of the film is controlled along with the exclusion or inclusion of the alarms related. The Reels program control manages the reels where the film rolls are mounted. Lastly, the *SM\_stretch* program is the core of this equipment module. Here the command to the motion control are processed and generated. All the commands for the operative modalities are activated and the outputs of the resulting movement of the axes are processed. Moreover, alarms are activated and input data are modified for the correct control by the drive. The *EM\_PD3\_Infeed* is dedicated to the control of the in-feed part of the machine and of the Layer axis. The two are grouped under the same equipment module because they share the same access to the products. Faults in one of the two parts affect the other. For example, a stuck package near the Layer axes generates a stop for both the actuator and the belt. This includes both the belt and the first part of the motion axes, namely the Layer. Firstly, the activation of the belt is controlled and the reference speeds are computed. This takes into account the state of the machine about which this program takes information from the upper modules. In addition, all the operative states of the Layer are controlled from here where the commands are activated.





Figure 1.3: Modules Programs.

## 1.3 Structure of the Schneider motion control

In this section, as mentioned before, the organization and composition of the Schneider motion control algorithm is described.

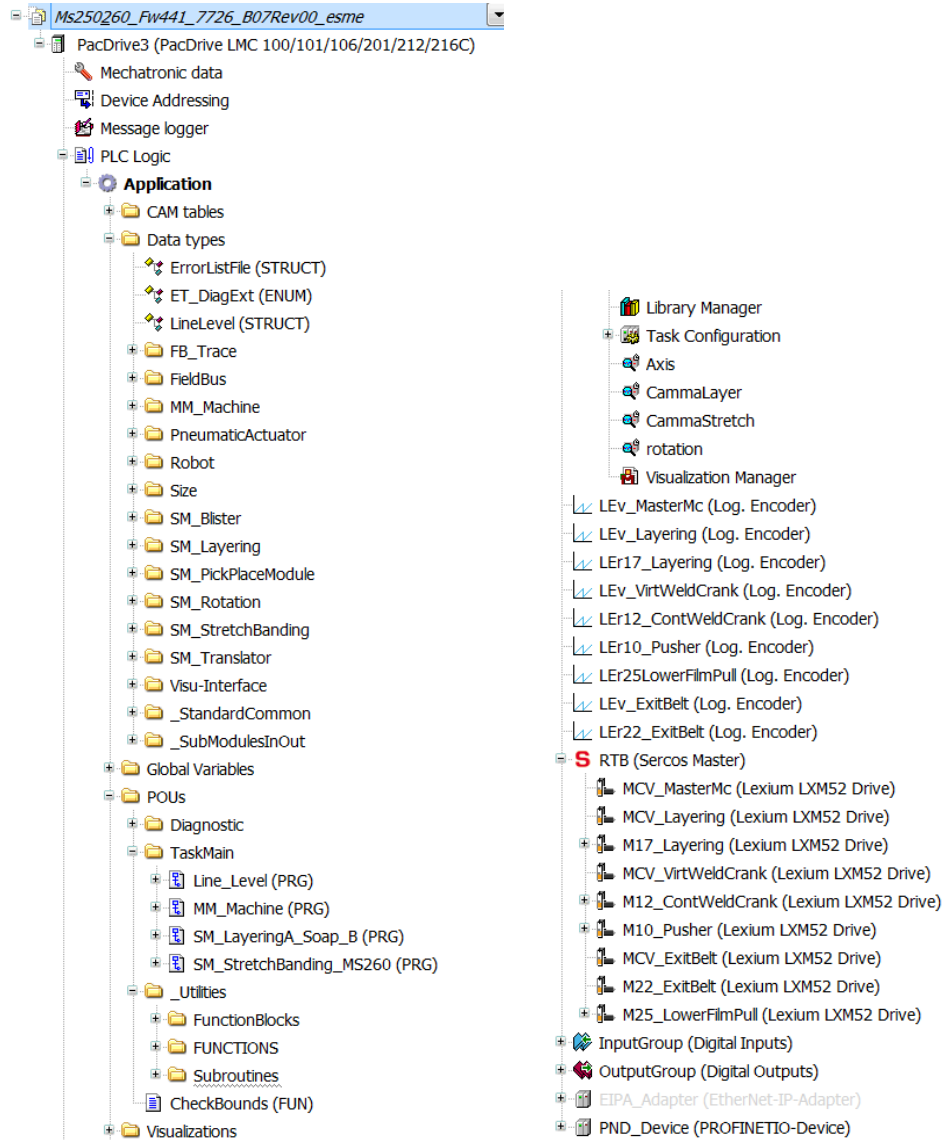


Figure 1.4: Schneider Tree.

First, Fig. 1.4 shows how the software is organized. It is divided in three sections. Regarding the first one, the variables and variable structures used inside the code are defined in the top. Then, the programs executed by the software are present. They represent the core part of the software that later will be analysed. At the end of the project, the *Utilities* folder which allows a better organization and execution of the code is placed. The second section contains the list of the hardware parts. In particular, for the correct functionality of Schneider, the motors encoders are defined for each virtual and real axis present in the machine. Below, the typologies of the motors are defined. The

last section regards the physical inputs and outputs of the drives. In this chapter, the programs, which will be substituted in the Siemens software, are analysed while the motor discussion is performed in the next chapter.

As Fig. 1.5 intuitively shows, there are four different programs generated in order to perform different roles of the motion control:

1. The **Line\_Level** is the communication line, inside the Schneider software, which has the role to establish, maintains and perform the connection and data exchange with the PLC. This POU will be analysed along with the relative PLC FBs in the dedicated section;
2. The **MM\_Machine** manages the information coming from the Line\_Level along with the feed-backs from the axis modules in order to maintain and manage the status of the motion part;
3. The **SM\_Layering** and the **SM\_StretchBanding** are the two sub-modules which convert the commands from the MM\_Machine into action directly controlling the axes.

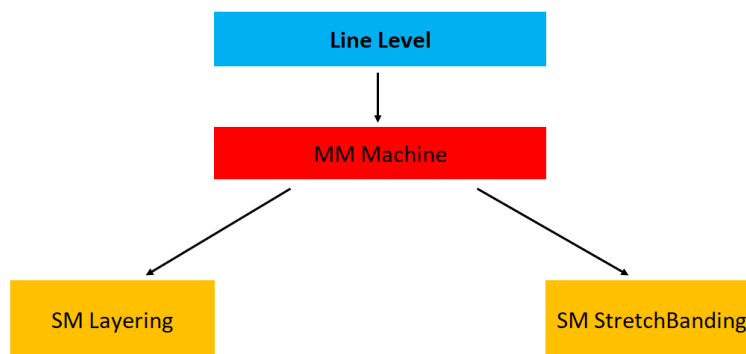


Figure 1.5: Schneider program structure.

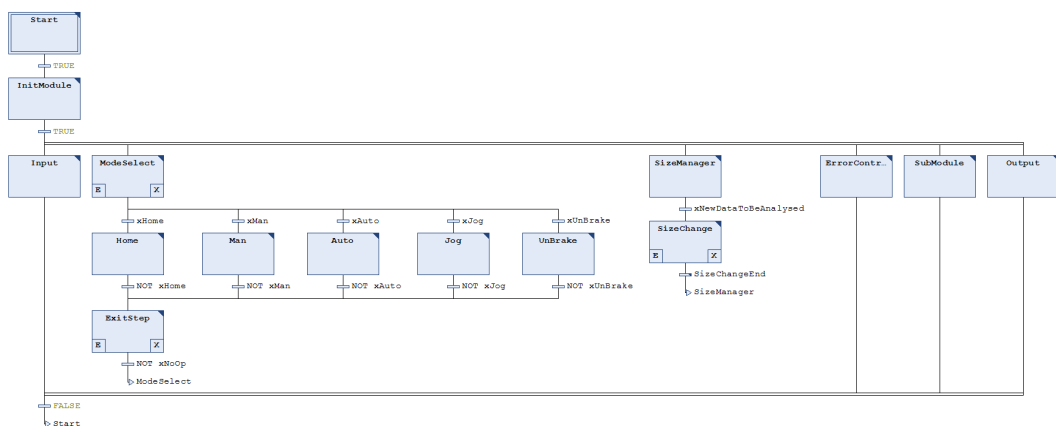


Figure 1.6: POUs Tree.

---

### 1.3.1 Machine module organization

The MM\_Machine and the sub-modules have the same structure that is shown in Fig. 1.6 on the previous page. In this way, it can be understood how the modules are organized. Outside the parallel action, only few steps are present. As soon as the machine starts running, they are executed in order to initialize their relative POU's. Afterwards, the Schneider motion control loops inside the parallel action.

At this point, different jobs are executed between the MM\_Machine and the sub-modules. Starting from the first one, it is in charge to control and manage the status of the Schneider software. In particular, it is the bridge between the Line\_Level, which communicates with the PLC, and the axes modules. This one manages also the actions carried out by the sub-modules. Indeed, the synchronization of the axes during the nominal operative conditions are automatically managed by the sub-modules as it will be explained later; but, this POU allows communicating to them how the operative conditions need to be executed. Talking about the structure, as it can be seen in the last Fig. mentioned, this module is mostly operating inside the parallel actions. The central part, namely the alternative sequence, is never used. Better to say, the module crosses them in parallel to the execution of the other blocks following the same logic that characterizes the management of the sub-module actions, but does not perform any type of operation. The important parts of this module are the blocks at the ends.

In the *Input*, the sub-modules underneath it are checked. This means that if any change is detected from their definition inside the Line\_Level, they are updated. Moreover, the bridge between the variables of the Line\_Level and of MM\_Machine is present. After that, the error reaction that the sub-modules must execute in case of faults is controlled. This coincides with the control of their enabling status. The details of the alarms handling is explained in a successive section.

Another important action is the *Format Change*. Inside this FB the size change procedure of the axes is started. Every time the updating procedure of the data in the HMI is completed and their validation in the PLC is successful, this module performs the first step of their management. In particular, the main data are processed in order to compile the new bundle dimensions. After this, the MM\_Machine waits for the sub-modules to complete their own procedure before ends its own.

At the end, likely the Input action also the Output represents a bridge between the locals and the global variables that will be accessed by the Line\_Level and the sub-modules.

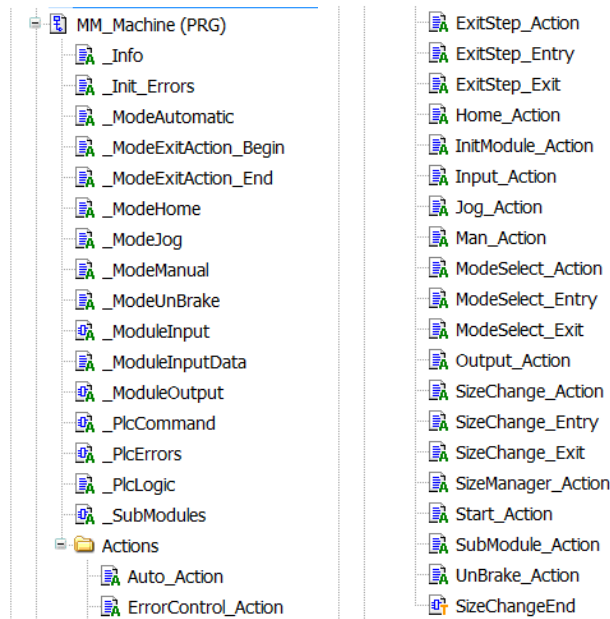


Figure 1.7: POU SFC Tree.

### 1.3.2 Sub-modules organization

The sub-modules perform similar instructions to the MM\_Machine. In particular, the tree is organized in: Module managing actions, Format change actions and Motion actions. The first one is the most similar between the two sub-modules as it contains the way in which the axes are controlled. This means that the communications actions with the upper modules, as long as the alarms managing, is the same. Then, the format change actions start to differentiate because the SM.StretchBanding has a more complex system to manage compared to the SM.Layering. This aspect is the reason why the last actions of the motions are more different.

#### Module managing actions

The Managing actions are divided in:

1. Input action
2. Error control action
3. AxisFB action
4. Output action

**Input Action** The Input action processes the input parameters. To enter more in detail, as a communication channel is established between the drive and the PLC, the information passing through it can change at any time. To allow the drive to understand when this happens, inside this action a FB checks whenever a variable alteration is performed, activating an updating procedure

---

of the variables. After this, the standard command inputs for the axes are copied from the MM\_Machine variables to the internal sub-module variables. Lastly, an update of the homing and the alarm reactions are carried out. In particular, for every axis inside the sub-module, the related *homeResetRq* is controlled at every cycle to activate the homing procedure whenever requested. At the same time, regarding the alarms generated from the axes, this action contains an instructions list that deactivates the module in case a certain error condition occurs. This part related to the alarms will be analysed later.

**Error control action** The Error control action is just a routine step that uploads the module alarms in a global library where they are processed and handled. The way this is carried out is discussed in a different section. Here, the *Global Exceptions list* is updated like it is in the two sub-modules.

**AxisFB action** In the AxisFB, complementary instructions to the input action are present. In particular, while the Input action activates the home requests for the axes, this step disables the bit because the request has already been accepted inside the same CPU cycle in which it was generated.

**Output action** The output action has multiple roles like the Input one. In particular, some control logic is executed here. Firstly, it has a filtering role. It updates the Drive and the internal variables by processing the status bit of the axes. Then, it counts the layers performed by the machine in order to commute, for the SM\_Layering, the axis cam. Instead, for the SM StretchBanding, this action analyzes the position of the axes in order to manage the synchronization bits. Moreover, it checks the online cam parameters of the Crank and of Film pull axes. In this way, the operator has the possibility to change the timing between the motors during nominal operations.

### **Format change action**

The Format change actions are performed every time a format change is requested from the operator. If this happens, the logic stops the motion control of the motors and activates the updating routine of the formats. In both the sub-modules, these actions take the parameters of the new package and bundle, prepared by the MM\_Machine module, in order to build the new cam profiles that the axes will follow. The same approach for the updating procedure is used in both the sub-modules. In the first action, the request of the format change, which was successfully taken by the PLC logic, is processed. In this way, the successive action is activated only in case of a correct update of the parameters. Next, the SizeChange starts the computation of the cams. Firstly, the new data are stored in the internal variables of the module. Then, the new bundle dimensions are computed. Accordingly, the new positions of the cams can be defined and set. At this point, the SM.StretchBanding generates a slightly more articulated algorithm in relation to the bundle dimensions. The Pusher and the Crank axes are characterized by different cams. Along with

---

this procedure, also other parameters can be modified, such as the homing positions or the speed limit of the gearboxes. At the end of the procedure, if the homing positions were changed, a new homing request can be raised.

### **Motion actions**

The core of the Schneider control is performed inside the actions of the alternative sequence. The motion of the two sub-modules is managed in the same way. The *ModeSelect* action allows the activation of one of the following actions: *Home*, *Man*, *Auto*, *Unbrake*. Then, whenever the execution of one of the previous steps is terminated, the motion continues in the *ExitStep*. Below, what these actions perform is explained. From Fig. 1.6 on page 18 it can be seen that also the action *Jog* is present. But, in this machine, this functionality is executed by the Manual action and only for the SM\_StretchBanding.

**ModeSelect** As the name suggests, the *ModeSelect* allows selecting one of the linked operations. This is achieved through a case that passes through the states if the previous condition is not fulfilled. In this way, if the initial condition to activate the motion is enabled, one of the actions can be activated.

**Home** The first block is performed in order to execute the homing of the axes. In the SM\_StretchBanding this step is more complex than the other because more axes are managed. Instead, the SM\_Layering is more simple because only two axes are present and the homing is straightforward. In particular, apart from the homing, also the cleaning procedure of the sealer is performed here. It consists of the positioning of the Crank axis in order to allow the cleaning of the sealing part of the machine. This procedure is performed in three successive phases after each of which the search for the axis reference position is always required. So, this procedure has a strict interaction with the homing. For this reason, it is executed inside the same action. Moreover, if the SM\_Layering homing is performed by activating the homing command for the axes, in the SM\_StretchBanding a longer case is present. Firstly, the Crank axis is controlled because it is connected to the motion of the pneumatic bundle presser. Then, also the other axes are homed if needed.

**Man** The second action of the alternative sequence is represented by the manual procedure. This type of operative condition is performed during the machine set-up, namely when the machine needs to be prepared. For this reason, the manual action allows a slower motion of the axes along with different modalities with respect to the automatic mode. In particular, in each sub-module two main modalities are present:

1. The autonomous motion with respect to the other modules of the axes;
2. The combination of modules motion through the Bypass mode.

---

For the SM\_Layering, in the first case, the axis is positioned through the cam selection and, therefore, the master control. This positioning allows setting the Layer axis in the two operative steps that it reaches during the nominal operations: the middle position, where the Layer fills the buffer, and the top position, where the bundle is pushed in its ready state.

Differently, in the SM\_StretchBanding the machine can execute two modalities. One allows to move only the Crank axis and executes the welding cycle. The second one controls all the axes synchronously like in nominal conditions, but this motion is controlled with the "Jog Button". In this way, the operations are performed slowly and through a Start&Stop control. This allows to understand at each angle of the master what is the position of every axis and to better define the cam positions. The second modality is the bypass mode. This is a particular condition where the layering axis is attached to the SM\_StretchBanding master through the bypass cam defined in the Format Change. Then, the motion is carried out by the same Start&Stop control. With respect to the previous one, in this case, the Crank remains in an idle position as it is also for the Film Pull axis. This modality allows the use of products that are not wrapped by the film and the exit bundle is characterized by only one layer.

**Auto** The nominal and automatic mode is performed inside the Auto action. This is the main state of the motion control. Indeed, under normal conditions, after the start-up, the machine reaches directly this step, making it the normal waiting step of the machine instead of the Select case as it can be thought. Like in the Manual action, inside this one, different modalities are present. One is the Bypass mode which is executed with the same conditions as before. The second is the nominal operative condition. For the SM\_StretchBanding, also the Sealer emptying modality is present. This one is carried out every time the sealing sensor detects an obstruction. Under such conditions, only the Crank axis is moved. Regarding the nominal mode, it is composed firstly of some synchronization steps for each sub-module. Then, the execution of the motion is performed. In the SM\_Layering, the master runs continuously and the Layer axis is directly attached to it whenever the conditions command it. The logic through which the cam of the Layer is selected is executed inside the Output action which is previously discussed. Here, only the conditions that allow the modification of the variables in that logic are present. As it can be understood, the only synchronization parameter with the other module is represented by the position of the Pusher axis. When it is in its back position ready to be loaded, the Layer performs its longest stroke. For the SM\_StretchBanding, the number of axes is much higher, which makes the control of each one separately more difficult. For this reason, the motors are directly coupled to the master at the beginning of the action. Then, only the master is controlled through Schneider controls in relation to the functionality of the Layer module. Indeed, if the layer has performed its longest stroke and it has loaded the prepared bundle, the master is allowed to perform an entire cycle. Otherwise, it stops waiting for the products.



---

**Unbrake** The last action is the unbrake. This is used only in case a motor is with a brake. In this machine, this is the case only for the Crank motor which, actually, controls a vertical mechanic system. Therefore, this action is executed only in case it is of interest to release the brake of the Crank system.

**ExitStep** When all of these actions are terminated, the logic passes through the Exit step. It allows to reset the axes operations or even to stop their motion in case one of the previous motion steps is terminated prematurely. Indeed, this step removes all the motion control commands and allows the start of the next motion step without operative conditions running. This step is very important in case an alarm, error, or emergency stop has occurred during the motion.

## 1.4 Schneider alarm managing

An important part of the Schneider motion control is the alarm generation and handling. This tool is provided by the software *FB\_ExceptionHandler* and the *FC\_SetException*. In particular, the code exploits these integrated functionalities in order to make the drive able to self-manage the errors and alarms which occur during the motion. This means that the alarms generated are added to the alarm list and are read in order to execute the required reactions. The functions used are called inside each of the modules of the motion control algorithm inside the *ErrorControl* action. In addition, each POU defines its own alarms in the initialization procedure while the machine is started.

To enter the details, the FB works in two different modalities on the base of the input variable *ixLocalExceptionHandler*:

1. If it is set to TRUE:  
the Error Handler is configured as a local management block to the sub-module and its function is to propagate towards the modules or sub-modules from it controlled the list of reactions it receives from the upper module.  
In this configuration the *iq\_stExceptionList* is not a source for generating reactions as the *istReaction* input is used. The *iq\_stExceptionList* is read by the module to get information about who has generated the error. In the event that the error comes from the module to which the BF belongs or from one of its sub-modules, the *qxError* output becomes TRUE and in *q\_udiSubModuleInException* we find the *moduleId* of the sub-module in error. Fig. 1.8 on the next page can be considered as reference.
2. If it is set to FALSE:  
In this case the *FB\_ExceptionHandler* is configured as a high level handler; its task is to acquire from the *iq\_stExceptionList* the errors that come there register and start the generation of the reactions to the other modules or sub-modules.

In this case, the block generates an internal list of reactions starting from the global error list which will then be transferred to the modules it controls. The ErrorHandler transfers the reactions to the sub-modules through the *astSubModuleInterface* structure whose address is passed to him in the mail (*i\_pstSubModuleInterface*, *i\_diSizeOfSubModuleInterface*, *i\_diNumberOfSubModules*). Fig. 1.9 can be considered as reference.

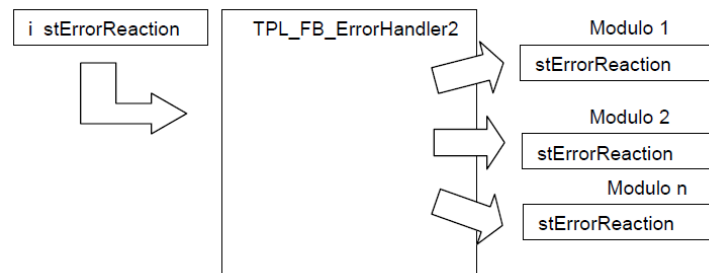


Figure 1.8: Error Handler acquisition.

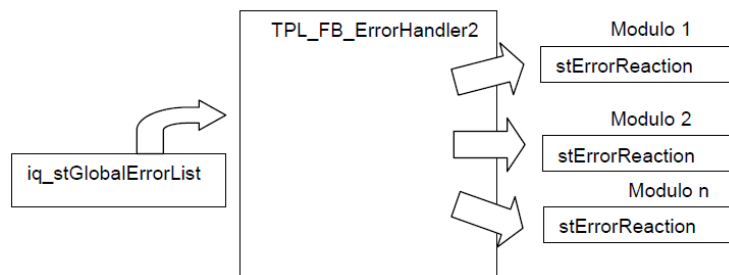


Figure 1.9: Error Handler transmission.

The mechanism of translation and transfer of reactions by table is realized through the *stErrorReactionTable*; for simplicity this can be seen as a cube whose dimensions are:

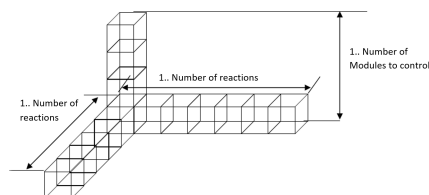


Figure 1.10: Reaction Table.

Each box must be filled with a TRUE or a FALSE in order to perform the translation errors. If we consider each plane of the cube associated with a module, each of these will result associated with a square matrix as follows:

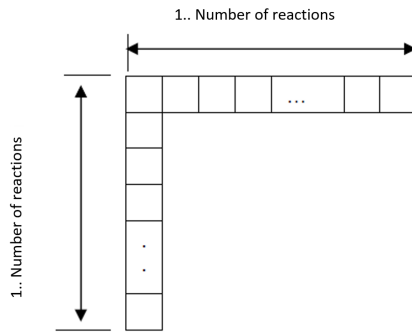


Figure 1.11: Reaction Table layer structure.

In order for the error translation to be carried out, the matrix must be initialized; to that purpose the library function *TPL\_FC\_InitErrorReactionTable* must be used. Thanks to the library function, each layer of the cube (therefore each square matrix relating to a form) is filled on its diagonal with TRUE while with FALSE in the others boxes.

T	F	F	F	...	F	F
F	T					
F		T				
F			T			
.				T		
.						
F						

Figure 1.12: Reaction Table: layer composition.

This means that the  $n^{\text{th}}$  error reaction entering the Error Handler will be transferred in the homonym error reaction directed to the module. Schematically it can be seen in Fig. 1.13 on the next page. If the assignment of the reactions must be changed, a TRUE or FALSE as Fig. 1.14 on the following page shows must be placed in the boxes in the matrix for each module with the appropriate reaction desired.

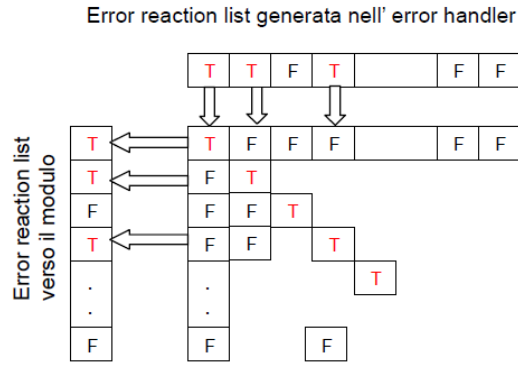


Figure 1.13: Reaction Table: layer meaning.

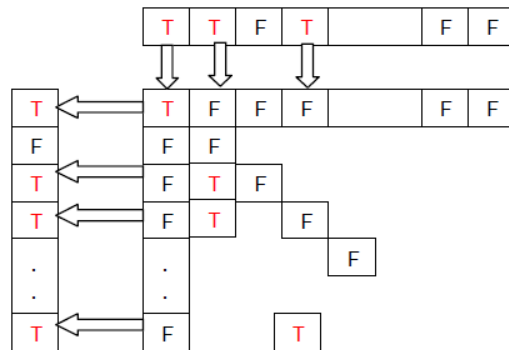


Figure 1.14: Reaction Table: layer modification.

Each module needs to generate its own errors. Indeed, the Schneider software allows to automatically manage the axes along with their drivers thanks to the software definition it is given to them. But, from the machine point of view, it is necessary to generate some errors useful for the correct functionality of the operations. To do this, it is sufficient to define the error in an initialization phase and use the *TPL\_FC\_SetError* function. This is practically done inside one of the initialization states which anticipate the parallel actions section. As Fig. 1.15 on the next page shows, for each error is defined which reaction is required for this error, the message displayed when it occurs and if it is reset automatically after its activation. The error definition is present in each of the POU's. Moreover, as it was explained in the Error Handler analysis, each of them also has the Error manager action. This is the ErrorControl action present in each SFC tree. Fig. 1.16 on the following page shows the function call needed in order to send to the Global Exception List the new error generated inside the module. This is valid for all the modules, sub-modules included.

```

stUserAlarm[1].stReaction.axReaction[uiER_Offset+TPL_ET_Reaction.StopEndOfCycle] := TRUE;
stUserAlarm[1].sMsg := 'GearBox speed too high';
stUserAlarm[1].timDelayTime := t#0ms;
stUserAlarm[1].wHmiAttributes := 0;
stUserAlarm[1].xAutoQuit := FALSE;

stUserAlarm[2].stReaction.axReaction[uiER_Offset+TPL_ET_Reaction.SyncStopEH] := TRUE;
stUserAlarm[2].sMsg := 'Immediate stop from PLC';
stUserAlarm[2].timDelayTime := t#0ms;
stUserAlarm[2].wHmiAttributes := 9999;
stUserAlarm[2].xAutoQuit := FALSE;

stUserAlarm[3].stReaction.axReaction[uiER_Offset+TPL_ET_Reaction.SyncStopEH] := TRUE;
stUserAlarm[3].sMsg := 'free';
stUserAlarm[3].timDelayTime := t#0ms;
stUserAlarm[3].wHmiAttributes := 0;
stUserAlarm[3].xAutoQuit := FALSE;

(* Initialization of the error reaction table *)
TPL.FC_InitReactionTranslationTable( i_udiNumberOfSubModules:= DINT_TO_UDINT(c_diNumberOfAxis),
                                     i_pstReactionTranslationTable:= ADR(astReactionTranslationTable));
(* Initialization of the error reaction translation table *)
TPL.FC_InitReactionTranslationMode( i_udiNumberOfSubModules:= DINT_TO_UDINT(c_diNumberOfAxis),
                                    i_petReactionTranslationMode := ADR(aetReactionTranslationMode));

```

Figure 1.15: Error definition example.

```

IF c_diNumberOfAlarms > 0
THEN
  FOR diAlarmsIndex:=1 TO c_diNumberOfAlarms
  DO
    IF axAlarm[diAlarmsIndex]
    OR stUserAlarm[diAlarmsIndex].xAutoQuit
    THEN
      TPL.FC_SetException(i_xExceptionActive:=axAlarm[diAlarmsIndex],
                          iq_stException:=stUserAlarm[diAlarmsIndex],
                          iq_stExceptionList:=stGlobalErrorList,
                          iq_stLogDataList:=stLogDataList);
    END_IF
  END_FOR
END_IF

IFB_ExceptionHandler(
  i_xEnable:= TRUE,
  i_xDiagQuit:= FALSE,
  i_stReaction:= iq_SubModuleInterface.i_stReaction,
  i_udiModuleId:= iq_SubModuleInterface.i_udiModuleId,
  i_sModuleName:= c_sModuleTypeName,
  i_pstSubModulesItf:= ADR(astSubModuleInterface),
  i_udiSizeOfSubModulesItf:= SIZEOF(astSubModuleInterface),
  i_udiNumberOfSubModules:= DINT_TO_UDINT(c_diNumberOfAxis),
  i_petReactionTranslationMode:= ADR(aetReactionTranslationMode),
  i_pstReactionTranslationTable:= ADR(astReactionTranslationTable),
  i_pstReactionTranslationJobs:= ADR(astReactionTranslationJobs),
  i_xLocalExceptionHandler:= TRUE,
  iq_stExceptionList:= stGlobalErrorList,
  iq_stLogDataList:= stLogDataList,
  q_xActive=> xErrorHandlerActive,
  q_xReady=> ,
  q_etDiag=> ,
  q_etDiagExt=> ,
  q_sMsg=> ,
  q_udiSubModuleInException=> udiErrorHandlerSubModuleInError);

```

Figure 1.16: Error handling example.

Anyway, some differences are present between the modules. The first one resides in the MM\_Machine. There, the reaction table is modified for the two sub-modules. Indeed, the reaction required for them is selected in case the error occurred in the other one. A second difference is the top module Line Level. Being the master module for the Schneider software, it contains the activation of the reaction transmission by reading and managing the Global Error List if at least one error generated in one of the modules is present. The only practical code is related to how the function is called, depending on the variable *ixLocalErrorHandler*.

To sum up, the alarms are generated inside each module. Then, they are added to the Global Exception List along with all of their details needed to identify the type of alarm. Then, the Line\_Level examines the global list and passes to the module/modules below it (in this case only the MM\_Machine) the relative reaction it needs to execute. This one passes the reaction required to its own sub-modules (SM\_Layering, SM\_StretchBanding) depending on the Reaction Table which it modifies at the beginning of each cycle. In this way, for each alarm generated in the modules, each sub-module can have its own reaction to be performed leaving an important d.o.f. when more sub-modules are present.

The Fig. below shows an example of this functionality. In this case, the dashed line represents the starting signal of the error generated in the sub-module. The code of the error is then processed in the *TPL\_FB\_ErrorHandler* which sends to each sub-module the correspondent reaction defined in the Reaction Table. This same process is then repeated in each sub-module which analyzes the received reaction to communicate it to its underlying modules through its table. This is executed until each module has received the reaction correctly.

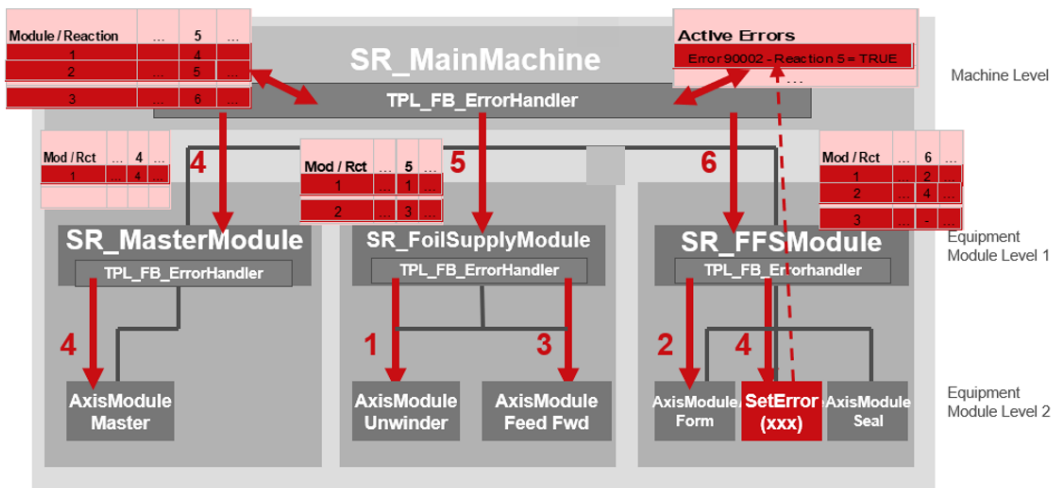


Figure 1.17: Example of exceptions transmission.

---

## 1.5 PLC-Drive connection

In this section how the communication is established and is performed between the PLC and the Drive is analysed. Considering that the software control is divided between two different domains, the Siemens TIA-Portal for the PLC status control and the Schneider Machine Expert for the motion control, it is necessary to have a dedicated part in both the algorithms to allow proper communication between the two worlds.

Inside the PLC, there is an entire OM dedicated to this topic, which is shown in Fig. 1.18. In particular, it contains, in addition to the program parts which makes the connection possible, the *Generale Axis V03* which acts as the manager of the variable communication with the motion control. Inside this FC, the PLC communicates to the motion software the conditions and the instructions to which the it must answer. That is, the ready, the stop and the modality that the drive activates. In addition to this, the communication parts present are:

1. the FCs which copy the variables to and from the drive (which are respectively *PN Plc-Axis\_V00* and *PN Axis-Plc\_V00*);
2. the FC which manages the format change variable transmission with the motion controller, *PN ChangeFormat Axis\_V00*;
3. the programs which check the node of communication, *PN Main\_V00* and *Profibus*.

There are also Data Blocks which contains all the variables used to pass and receive the information with the Drive.

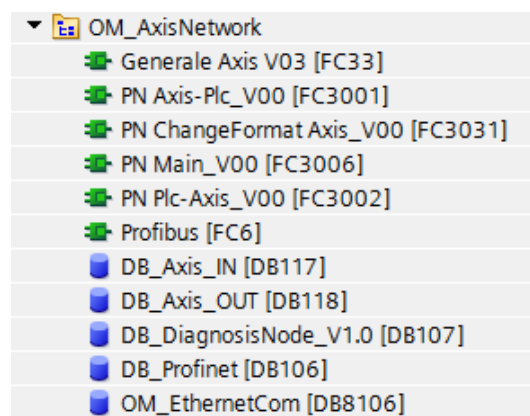


Figure 1.18: OM structure.

Inside the Drive, the POU *Line\_Level* previously mentioned in the Schneider section is fully dedicated to communication managing. This one has a different structure with respect to the other modules of the motion control. As Fig. 1.19 on the following page shows, it has fewer actions and a simpler structure. Only some of them are of interest. The *ReadIn* action performs a variables update. This concerns the updating of the input main variables from

the *DB\_Axis\_In* of the PLC, the updating of the alarm reaction that this module manages, the updating of the format variables coming from the PLC and reading of the format-change conditions of the MM\_Machine. Next, the *ErrorControl* executes the alarm control previously discussed in the above section. Lastly, the *WriteOut* updates the internal drive variables to the *DB\_Axis\_Out* data block of the PLC along with all the information related to any error that occurred during the last scan.

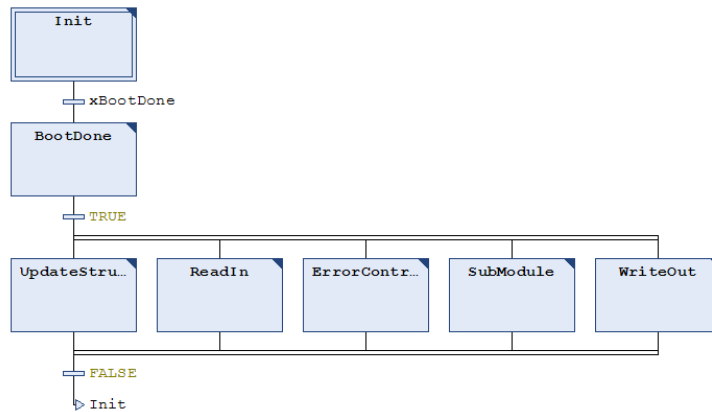


Figure 1.19: Line Level structure.

After the software parts have been described, how the communication occurs is explained. Firstly, it is necessary to pass the data between the two software. This is performed with Words transmission along the Profinet network. Indeed, the data blocks are reorganised in Words variables. They are in this way moved from the PLC to the drive and the other way around. These operations are performed by *PN Plc-Axis\_V00* and *PN Axis-Plc\_V00* in the PLC and in the *ReadIn* and *ReadOut* for the drive.

*The field bus real time command and data are checked each cycle*

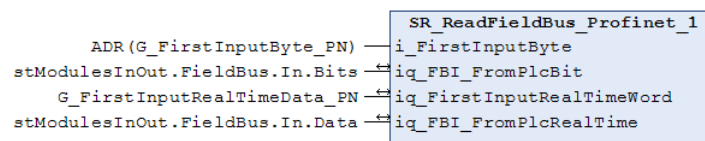


Figure 1.20: Data Transmission Schneider.



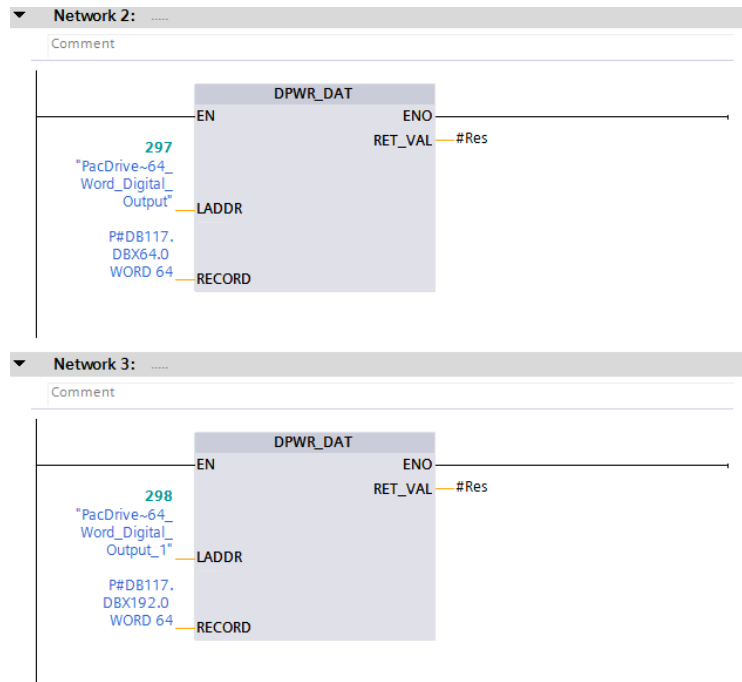


Figure 1.21: Data Transmission PLC.

Along with this, it is also necessary to control at every time if the connection is working. The communication is established as soon as the machine is started. From that instant on, a vitality-check bit is managed. This bit passes through the two software and is modified at each cycle. Simply, the bit is raised at the beginning of the initialization procedure. It is transferred to the motion control which copies it again without modification inside a different PLC variable. This last one allows checking if the connection is present. This is because if the bit returned by Schneider is not TRUE for a sufficient amount of time, an alarm is activated. Furthermore, these steps allow checking only if the connection is successfully established at the beginning. To understand if the connection still works, the PLC deactivates the initial bit only in the case the returned one is TRUE. In this way, after the Machine Expert software resets its own variable passed to the PLC, the process can be repeated. It comes with itself that if the motion control does not reset the bit as the PLC did, the alarm is triggered analogously. This simple but fundamental procedure allows understanding if at any time the connection between the two worlds is active. This is only one of the aspects that will be considered in the conclusions with respect to the alternative motion control developed.

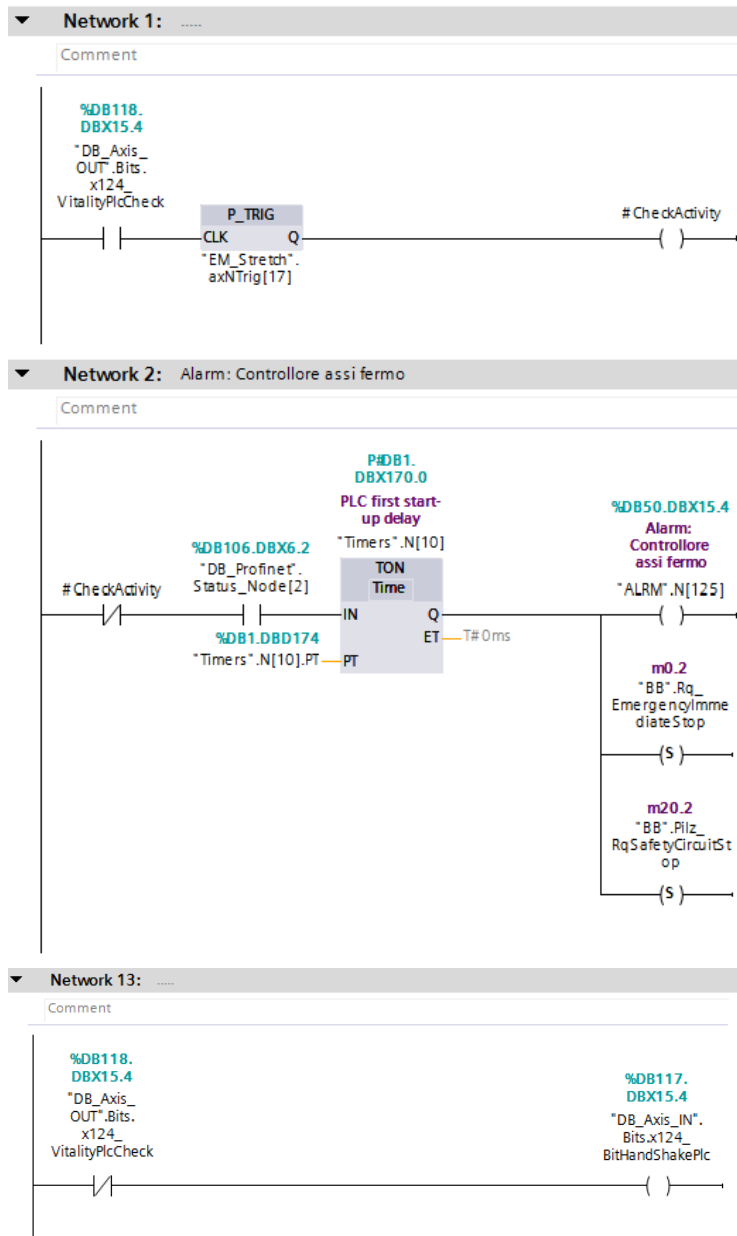


Figure 1.22: Vitality-Check of PLC.

# Chapter 2

## Hardware structure

In this chapter, the hardware structure of the machine is analysed in all of its aspects. To start, the Schneider part is presented. To better understand it, how the Schneider motor catalogue is organised is explained in the first section. Therefore, the hardware configuration mounted in the MS260 is presented. This contains both the actuators and the motion control as well as the composition of the mechanical systems that they actuate. At this point, the tests performed during the machine operations are examined which represent an important point in the comprehension of the motor mounted. Lastly, the Siemens world is considered. As it is for Schneider, the actuator catalogue is presented. Then, the considerations of the new alternatives are analysed. These contain both the presentation of the tools available in order to perform such analysis and the discussion of the possible solutions.

### 2.1 Schneider catalogue

The Lexium servo drive and the motors portfolio include a variety of ranges with a power range up to 24 kW for independent or synchronized motion control. Lexium products can adapt to demand high performance, power and simplicity of use in motion control applications. The Lexium Drives come in three different typologies: Lexium 32, Lexium 52, Lexium 62.

1. The Lexium 32 range of servo drives includes four servo drive models with power ratings from 0.15 to 11 kW and the servo motor ranges BMH and BSH series. This is the lower level of Schneider catalogue;
2. The Lexium 52 are Stand alone servo drives from 0.4 and 7 kW for PacDrive based automation solutions. In a conventional stand-alone design with an integrated 3-phase power supply, Lexium 52 series servo drives are particularly well suited for the economical configuration of servo drive solutions with self-contained single axes. The drives communicate via Sercos with the PacDrive 3 controller and offer embedded digital I/O. They are available in five different power levels, ranging from 1.5 to 24 A continuous current and 6 to 72 A peak current;

- 
3. The Lexium 62 are multi axis servo drives and servo motors from 0,95 to 24 kW for PacDrive based automation solutions. These are the Schneider solution for the increasing control cabinet space requirements and rising costs for mounting and cabling. This represents the better solution for machines with a large number of axes to control, which is not the case of the MS260. In any case, some features are here reported.

The Lexium 62 series servo drives consist of single drives (1 axis) and double drives (2 axes) of the same size. All of the single and double drives within a group share a single power supply. This multi-axis approach Lexium 62 is decreasing the required cabinet space and the installation times in a sustainable manner.

Dynamic, highly efficient servo motors form the basis for servo solutions with Lexium 52 and 62. The Lexium SH3, MH3, and the SHS stainless steel servo motors cover a wide range of performance and flange sizes. All motors are equipped with electronic nameplates, which is making them an integral component of the PacDrive 3 automation system, too.

Lexium SH3 servo motors are a solution for dynamic performance, providing a continuous stall torque range from 0.2 to 94.4 Nm for speeds up to 9,000 rpm. Lexium SH3 servo motors are more compact and offer a higher power density than conventional servo motors thanks to their new winding technology.

Lexium MH3 servo motors provide excellent power density values to meet the requirements of compact machines. With four flange sizes and three different lengths for each flange size, they are suitable for many applications, covering a continuous stall range from 1.4 to 65 Nm for speeds up to 6,000 rpm. The Lexium MH3 servo motors have a medium inertia motor, which means they are particularly suitable for high-load applications.

Lexium SHS servo motors are stainless steel servo motors that are based on Lexium SH3 servo motors and designed for a high-torque output with relatively low current consumption in a stainless steel jacket. Lexium SHS stainless steel servo motors and hybrid cable are the ideal choices to meet the requirements for dynamics, precision, and surroundings in the food and pharma industries.

## 2.2 MS260 motors description

It is important now to describe and analyse the motors present in the machine. This is necessary to better understand the requirements of the machine and to identify the limits of the actuators.

Among the possibilities shown in the previous section, the brushless mounted on the machine are part of the Lexium 52 servomotors and in particular of the SH3 family. In this category, the possibilities of servo-motors are limited. On one side, Schneider offers the actuators needed to cover a wide range. On the other, the degrees of freedom for the choice of the right motor are decreased by the small number of alternatives as the flange sizes of the actuators are only of 6 types. This is going to be one of the points of comparison with respect to the Siemens offers. In particular, the axes are moved thanks to two different types of brushless motors: SH30702P11A2000 and SH31002P11F2000. To better understand the names, Fig. 2.1 shows what each term means.

Lexium SH3 servo motors – References										
To order a Lexium SH3 servo motor, complete each reference with:										
	SH3	***	*	*	*	*	*	*	**	
Flange Size	40 mm (1.575 in.)	040								
	55 mm (2.165 in.)	055								
	70 mm (2.756 in.)	070								
	100 mm (3.937 in.)	100								
	140 mm (5.512 in.)	140								
	205 mm (8.071 in.)	205								
Stack Length	One stack (all flange sizes)	1								
	Two stacks (all flange sizes)	2								
	Three stacks (flange size 70, 100, 140, 205 only)	3								
	Four stacks (flange size 100, 140 only)	4								
Winding	Medium speed (480 VAC) (1)		P							
	Low speed, current optimized (2)		M							
Shaft end	Smooth shaft		0							
	Keyed shaft		1							
Encoder system	Absolute SinCos Single Turn - 128 traces per turn (SKS 36)				1					
	Absolute SinCos Multi Turn - 128 traces per turn (SKM 36)				2					
	Absolute SinCos Single Turn - 16 traces per turn (SEK 37)				6					
	Absolute SinCos Multi Turn - 16 traces per turn (SEL 37)				7					
	One-Motor Cable connectivity	Absolute Single Turn (Hiperface® DSL 18 bit EKS36) (3)				A				
		Absolute Multi Turn (Hiperface® DSL 18 bit EKM36) (3)				B				
Holding brake	Without brake					A				
	With brake					F				
Connection	Two-Motor Cable connectivity	Straight connectors					1			
		Angular connectors					2			
	One-Motor Cable connectivity (DSL cable + Quick lock connector)	Straight connector (3)						3		
		Angular turnable connector						4		
Degree of protection	Shaft IP54 without shaft sealing ring, housing IP65, convection							0		
	Shaft IP65 with shaft sealing ring, housing IP65, convection							1		
Motor type	Standard: 00/Custom versions: 01...99								00	

Figure 2.1: SH3 servo-motors nomenclature.

So from this, it is possible to understand that the SH3070 is a 70 mm shaft motor with an Absolute Single Turn encoder. Moreover, it is without the holding brake and with a key on the output shaft. The SH3100, instead, is characterized by a 100 mm flange, a holding brake, the same encoder as the previous one and also a keyed shaft.

Accordingly to the machine described in the first chapter, there are four different motors that allow the functionality of the machine. Three of them,

---

the Layering, the Pusher and the Film Pull, are moved thanks to the smaller of the two actuators defined, the SH3070. On the other hand, the Crank is actuated by SH3100. The reason behind this choice is related to a practical one. Indeed, to maintain the costs low and reduce the number of differences of the machine hardware components, the three jobs are made with the same size of the actuator. The larger actuator, instead, needs to move a crank system. For this reason, the larger one is needed.

Gearboxes are paired with the actuators. In this case, as it is for the motors, two typologies are present: Neugart PLE and Neugart PLFE. The first one is a planetary gearbox. It is a very light product and at the same time extremely performing, also ideal for heavy production cycles thanks to the design with low friction bearing and optimized lubrication. It is paired with the Layer and the Pusher axes because they need to run across long distances. In addition to this, they are characterized also by a belt each which allows transforming the rotational motion of the gearbox output shaft into longitudinal motion of the piston. Differently, the sealing part is performed through a heavier system, the crank. The PLFE gearbox is a planetary gearbox with a compact flanged output shaft. You save more than a third of the space with a decidedly torsional stiffness higher in order to support the higher torques of the system. Instead, for the Film Pull, the motor is directly attached to the output shaft which actually comes into contact with the film.

In relation to the characteristics, the table below shows how these components are mechanically built. It can be seen that the Layer axis has a gearbox with a ratio of 1/7 and a belt with a ratio of 280 mm/rev. The last one means that the piston translates of 280 mm per revolution of the output shaft of the gearbox. The Pusher axis has a gearbox with a ratio of 1/7 as the Layer, but it has a shorted belt ratio which is only 200 mm/rev. The Film Pull has only a gearbox of ratio 18/30 as it has no belt. Lastly, the Crank has a two-stage reducer with a ratio of 1/20. As any belt is present in this case, the transformation between the output gearbox and the longitudinal translation of the crank is performed manually inside the software. This will be an object of discussion in the next chapter.

	Layer	Pusher	Crank	Film Pull
gearbox (GearIn/GearOut)	1/7	1/7	1/20	18/30
Motion conversion type	belt	belt	crank	none
Motion conversion ratio	200	280	360	360

---

## 2.3 Siemens hardware

In this section, how the possible alternatives to the Schneider actuators are analysed. In particular, how Siemens organizes its actuators is shown in order to better understand how the analysis of the possible substitutions is made.

### Siemens motors catalogue

To start, it is discussed how the Siemens motor catalogue is articulated. It offers a large variety of actuators for every use and situation. In addition to this, each category allows selecting different sizes of the motor in order to perfectly fulfil the requirements desired.

1. SIMOTICS S - Servomotors for exact positioning and high dynamics as well as for precise motion control.

Whether for positioning in pick-and-place applications or for cyclic drives in packaging machines or for continuous path control in handling devices and machine tools. Wherever highly dynamic and precise motion sequences are required, high-efficiency, permanent magnet-excited SIMOTICS servo motors are the best choice. Depending on the application, they are available with various integrated transducers - from simple resolver up to absolute high resolution transducer. SIMOTICS S motors are also optionally available with gearboxes.

2. SIMOTICS M - Main motors for regularity of rotation of rotary axes and main drives.

These are the main motors for applications that mainly require continuous and precise axis rotation. With a power range from 2.8 to 1340 kW, they cover virtually every possible application. Therefore, they are suitable as main drives for presses, roller drives for printing machines, textiles, paper and plastic processing machines. They are also used as drives for winders as well as in machine tool spindles and lifting devices.

3. SIMOTICS T - Torque motors for the direct and gearless drive of rotary axes.

The torque motors are optimized for high torques with low rated revolutions. Thanks to high precision and dynamics as well as low wear (absence of mechanical transmission elements) they are ideal as integrated motors for rotary table transfers, rotary tables or swivel and rotary axes, eg. for machine tools. This also applies to complete torque motors, which are used, for example, as roller drives or winders in converting applications.

4. SIMOTICS L - Linear motors for maximum dynamics and precision of linear movements

These linear motors are the ideal solution when linear movements must be performed with maximum dynamics and precision. The reason: the effects of elasticity, play and friction as well as natural oscillations in

---

the kinematic chain are largely avoided since the use of linear motors eliminates mechanical transmission elements such as ball screws, joints and belts. This simplifies the design of the machine and reduces wear.

5. Customized solutions for specific applications

Application-specific solutions could be needed. Therefore, Siemens allows to design and implement application-specific motor solutions perfectly tailored to the design and the performance required.

6. System solutions perfectly matched to each other

The SIMOTICS motors are perfectly matched to the drive systems of the SINAMICS family. Thanks to the integrated encoder with redundant traces and the safety functions integrated into the drive, modern safety concepts can easily be implemented. The use of additional external safety components is therefore completely unnecessary.

From this, it is possible to understand the large number of possibilities that the Siemens catalogue offers. Moreover, in each category, there are very different actuators that allow executing the same task with different alternatives so as not to limit the customer.

For the purpose of the MS260 project, the SIMOTICS S is the family to which take reference. It contains different sub-categories: 1FT7, 1FK2, 1FK7, 1FL6. The first ones are servo motors are permanent magnet synchronous motors with very compact dimensions and attractive design. They can be distinguished into two categories: the Compact category, thanks to their low torque ripple, are predestined for use on machine tools that must ensure a very high quality finish and excellent machining results. Thanks to their compact design, they find a place even in confined installation spaces. The High dynamic motors have very low rotor inertia to allow extremely high dynamics and very short cycle times. These motors are available with forced ventilation and water cooling types, so they are capable of delivering considerable power in continuous service. The second class contains compact synchronous motors with permanent magnet excitation. The motors are sized for operation without external ventilation and dissipate heat through their external surface. SIMOTICS S-1FK2 motors are characterized by high power density, degree of protection and overload-ability. The third motors are compact synchronous motors with permanent magnet excitation. With the available options, gearboxes and encoders as well as the expanded product range, the SIMOTICS S-1FK7 motors can be perfectly adapted to any application. They therefore also meet the ever-increasing requirements of new machine generations. Together with the SINAMICS S120 drive system, the SIMOTICS S-1FK7 motors form a powerful system with high functionality. The integrated transducer systems for the regulation of the n. of turns and position can be chosen according to the application. The last one SIMOTICS S-1FL6 servomotors are permanent-magnet synchronous motors and designed for operation without external cooling. The heat is dissipated through the motor surface. They have a 300% overload capability and can be combined with the SINAMICS V90 servo



drives to create a powerful servo system with high functionality. Incremental or absolute encoders can be selected depending on the application. SIMOTICS S-1FL6 motors have a high degree of dynamic performance, wide speed control range and high shaft end and flange precision.

## Motors selection tools

Under these circumstances, the stand-alone choice of motors is very difficult. For this reason, Siemens offers two different methods to size and select the right actuator.

**DT Configurator** The first one is on an online configurator called *DT Configurator*. It allows identifying the Siemens servo-motor that fulfil the requirement. But, in this case, the ideas related to the motor must be very detailed regarding the type and dimension of the motor. With respect to the second tool that will be presented, in this case, the first things that must be specified are the class of the motor among those defined above and also the sub-class, as Fig. 2.2 shows.

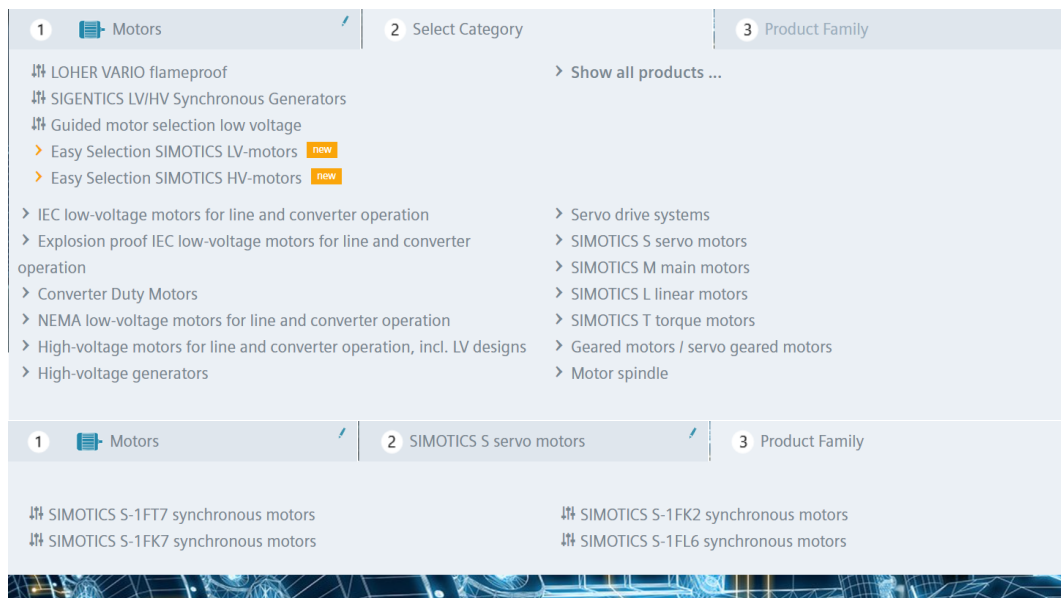
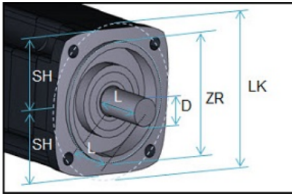


Figure 2.2: DT Configurator Initialization.

Then, as 2.3 on the next page shows, all the specific parameters of the motor must be added. At this point, the configurator shows, on the top, the name of the motor as the parameters are selected and, on the bottom, the alternatives that fulfil the specifics among the servo-motors on the catalogue.



Servomotor without gearbox: mechanical interface						
Article No. Selection	Shaft height SH [mm]		Centering edge ZR Diameter [mm]	Hole circle LK [mm]	Shaft extension D x L	
	nominal	exact			Standard	Alternative
1FK701x (i)	20	20	30	46	8x18	
1FK702x (i)	28	27.5	40	63	9x20	
1FK703x (i)	36	36	60	75	14x30	11x23
1FK704x (i)	48	48	80	100	19x40	14x30
1FK706x (i)	63	63	110	130	24x50	19x40
1FK708x (i)	80	77.5	130	165	32x58	24x50
1FK710x (i)	100	96	180	215	38x80	32x58

**Converter system**

Converter / voltage level

Line Module

**Selection**

Shaft height (from - to)

Type of cooling  Forced ventilation  Natural cooling

Static torque

Maximum (temporary) speed  $0 \leq$    $\leq 10000$  rpm

Holding brake

**Encoder system**

Drive-CLIQ  Yes  No

Device version

Resolution

Encoder

**Shaft extension**

Feather key  Yes  No

Shaft version

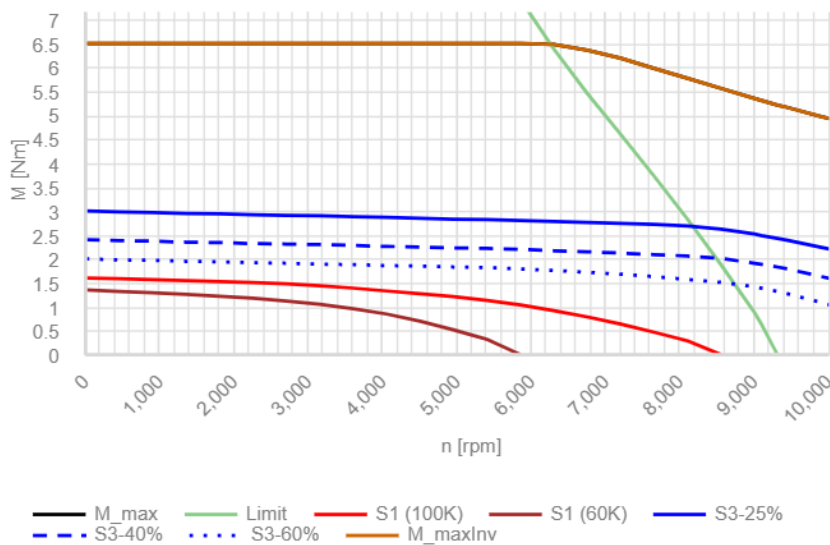
Shaft end D x Length

**Degree of protection**

Degree of protection

Figure 2.3: DT Configurator Parameters.

For example, it is inserted the motor 1FK7034-2AK71 which is one of those that will be discussed later and in Fig. 2.4 can be seen the alternatives proposed with all the main characteristics along with the speed-torque curves in the different operative conditions for the one selected.



Attention: The motor characteristic refers to the basic motor without additional options such as explosion protection, gearbox attachments, etc.

Standard motors

MLFB group, 1st - 1	Shaft height	Static torque	Rated torque	Rated speed	Max. torque	Max. speed	Moment of inertia
<input type="radio"/> 1FK70322AK7	36	1.10 Nm	0.80 Nm	6000 rpm	4.50 Nm	10000 rpm	0.0000650 kgm <sup>2</sup>
<input type="radio"/> 1FK70334CK7	36	1.30 Nm	0.90 Nm	6000 rpm	4.30 Nm	10000 rpm	0.0000250 kgm <sup>2</sup>
<input checked="" type="radio"/> 1FK70342AK7	36	1.60 Nm	1.00 Nm	6000 rpm	6.50 Nm	10000 rpm	0.0000900 kgm <sup>2</sup>

Figure 2.4: DT Configurator Alternatives.

**SIZER for Siemens Drives** The second method available is more advanced. It is a downloadable tool called *SIZER for Siemens Drives* which allows a more articulated selection of motors considering different aspects of the mechanisms where it will be mounted. To start off, the SIZER asks to create a project. Then, a new drive system must be added. This allows generating a new axis that can be sized. This comes with the definition of the properties of the new system that must be modelled. From Fig. 2.5, it can be seen how the specifications are articulated. The system must be defined by the type of drive system. For this project, the choice is a multi-axis system. Then, the type of module and lastly the motor family are identified along with the type of load it must move.

After the initial properties are set, the motor must be found. To do so, a window guides through the different steps. Initially, the load data are inserted defining the nominal torque that is necessary along with the maximum number of rpm. Also, the maximum torque can be added along with some other points of interest in the curve of the load. Then, some characteristics of the physical motor are also added such as the type of the encoder, the cooling and the presence of the brake. At this point, in the third step, all the compatible actuators are shown. Here, they can be filtered or ordered dependently on the types of torques, dimensions and speeds. Finally, the motor curves are represented in a speed-torque graph with respect to the points specified in the first point. These steps can be repeated until a motor is selected.

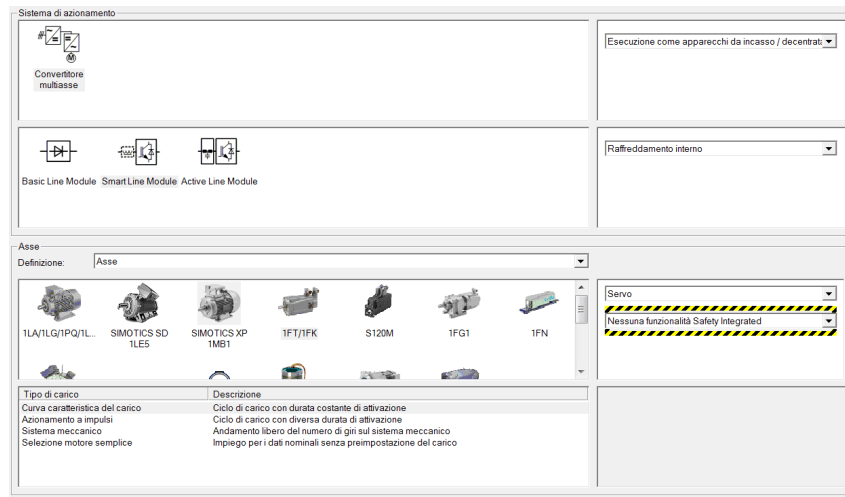


Figure 2.5: SIZER Initialization.

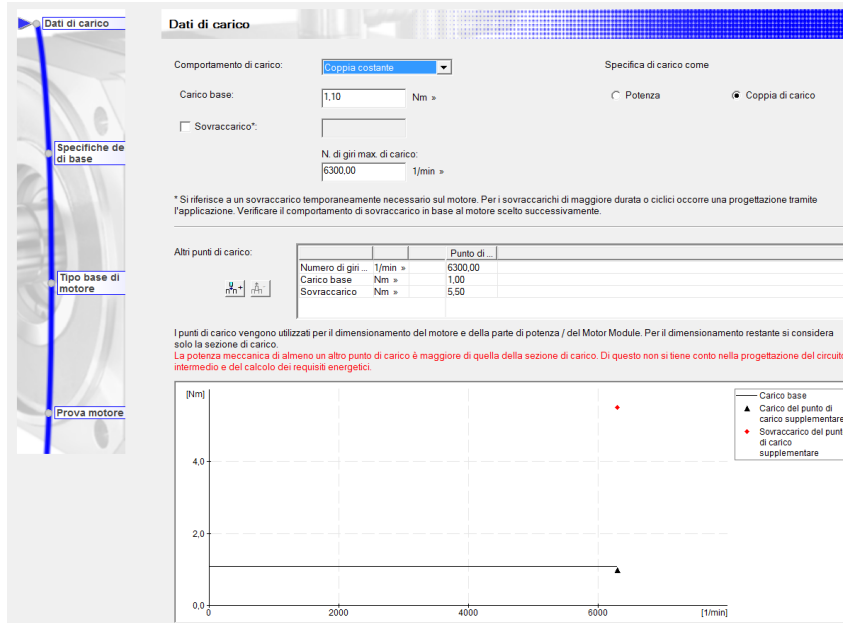


Figure 2.6: SIZER parametrization.

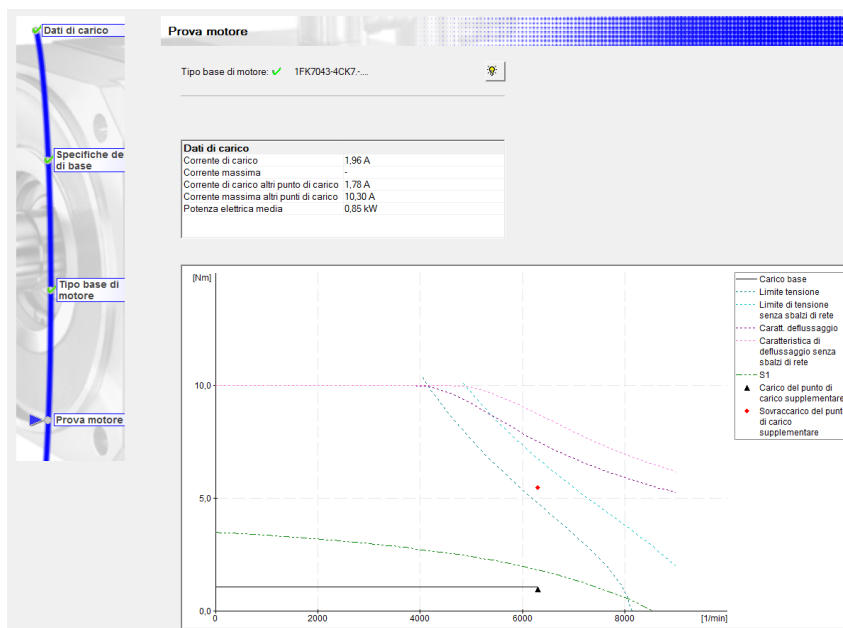


Figure 2.7: SIZER Motor Characteristics Curves.

---

## 2.4 Siemens PLC

Regarding the PLC, the version of the MS260 analysed is characterized by the ET 200 SP controller. It allows the execution of basic functionalities in terms of motion control. Indeed, it is performed by the Schneider drives and the PLC has only a role of supervisor. When it comes to the execution of the control of the whole machine, a new PLC must be selected. The one used is the upgraded version of the previous one: ET 200 SP Open Controller. It is the first controller of this type to combine the functions of a PC-based software controller with visualization, PC applications, and central I/Os in a compact device. The installed and preconfigured SIMATIC S7-1500 Software Controller, the PC-based version of SIMATIC S7-1500, is used for control. This controller is operated independently of Windows and thus ensures high system availability. It facilitates the controller's rapid start-up and supports Windows updates and reboot during ongoing operation. Even a failure of Windows does not impact the controller.



Figure 2.8: ET 200 SP Open Controller.

It is also possible to use different types of PLCs as Fig. 2.9 on the following page shows. In particular, those belonging to the family of the technology CPU. They are more dedicated to the execution of advanced motion control commands. Indeed, the Siemens PLCs for this purpose are classified accordingly to the number of axes and components that they can manage. Among the class mentioned above, PLCs for large and small machines are both available. However, for the machine under analysis, they are excessively or too less powerful. Namely, the components that they can handle are not in line with the ones needed. The Open Controller, instead, meets the requirements without exceeding too much from the boundaries. Moreover, the computation speeds, if compared, show that the former is near the performances of the larger CPU of the technology family. Lastly, the Windows dedicated part can be exploited for future developments.







Technology CPU					Standard CPU	Open Controller
CPU types	1511TF-1 PN	1515TF-2 PN	1516TF-3 PN/DP	1517TF-3 PN/DP	1518F-4 PN/DP (MFP) <sup>1</sup>	1515SP PC2 TF PN
Interfaces						
Program memory	225/225 KB	750/750 KB	1,5/1,5 MB	3/3 MB	4/6 MB	1/1,5 MB
Data memory	1 MB	3 MB	5 MB	8 MB	20 MB <sup>1</sup>	5 MB
Bit performance	60 ns	30 ns	10 ns	2 ns	1 ns	10 ns
Functions	Display, S7-1500 backplane bus					
Positioning axes						
• Typical <sup>2</sup>	5	7	55	70	128	30
• Maximum <sup>3</sup>	10	30	80	128	128	30
Motion Control Resources <sup>4</sup>	800	2.400	6.400	10.240	10.240	2.400
Extended Motion Control Resources <sup>5</sup>	40	120	192	256	–	120

Figure 2.9: Advanced controllers.

## 2.5 Siemens motor selection

In order to better understand the functionality of the machine and to find the correct alternatives for the actuators, I performed some tests. In particular, they have the purpose of knowing the functioning of the actuators during the operating conditions. This machine provides, for the formats stored in the PLC memory, two operational speeds: 24 ups/min and 60 ups/min. So, I performed the tests while the machine was running in both scenarios. Moreover, as the machine allows it and because it is more convenient, I considered the case of a No-Load-Run, which means that the machine runs without the presence of products. A test with products was performed in the case of 24 ups/min but there are not many differences with respect to the No-Load-Run. The products considered for the tests are 45x45x80 packages which need to be grouped into bundles of two layers and five products per layer. The Figures below show the results of the tests performed. In particular, the values on the "y" axes are in units. These units are those used to measure the type of motion of the axis moved by the motor-gearbox system. For a linear motion of the axis, the units are in millimetres, while for rotating one it is in degrees. This concept is applied for all the motion traces: position, velocity, acceleration and tracking error. Therefore, they are not be considered always in motor degrees. Instead, the current is measured in milliampere. On the X axis, the time is present.

The traces showed are:

- Blue traces: the position trajectory
- Green traces: the velocity profile
- Light green traces: the acceleration profile
- Grey traces: current usage by the motor
- Red traces: tracking error

From Fig. 2.10 on page 48 to Fig. 2.16 on page 50, the dynamic characteristics of the motors during the 24 ups/min scenario are shown. Instead, from Fig. 2.17 on page 50 to Fig. 2.23 on page 52 the 60 ups/min scenario is shown where the acceleration of the axes replaces the tracking error. I took these traces during the motion of the machine thanks to the Schneider software which allows monitoring the online execution of the processes. From these Figures, I made some considerations in order to understand the limits of operation of the actuators. In particular, it is possible to identify the displacement that each axis travels. In the case of the Layer, it is characterized by two different strokes. The longest is related to the push of the layers out of the buffer to the waiting position in front of the Pusher. The shortest, instead, is required to load the buffer. In Figures 2.11 on page 48 and 2.18 on page 50, two of them are present, as the bundle used for the tests is composed only of two layers. Then, one is used to load the first layer in the buffer and the other is used to push the whole bundle out (last and first layer). The other three axes have only one stroke each, characterized by the work they perform.

From the considerations on the position and the velocity, from the mechanical composition of the motions and from the analysis of the current traces, it is possible to determine the maximum and nominal values of both the velocities and the torques of each motor. The table below shows the result of such considerations.

1. 24 u/min and 55% speed of Layer axis;

	Layer	Pusher	Crank	Film Pull
Stroke Long	147	485	64	116
Maximum Torque (Nm)	2.31	1.54	2.541	0.308
Maximum Speed (rpm)	2100	2835	1400	411
Nominal Torque (Nm)	0.85	0.329	0.78	0.12
Nominal Speed (rpm)	747	942	397	64.5
Stroke short	68	–	–	–
Nominal Torque (Nm)	0.55	–	–	–
Nominal Speed (rpm)	747	–	–	–

2. 60 u/min and 75% speed of Layer axis.

	Layer	Pusher	Crank	Film Pull
Stroke Long	147	335	64	116
Maximum Torque (Nm)	4.389	5.39	15.73	0.77
Maximum Speed (rpm)	2940	6300	3333	833
Nominal Torque (Nm)	1.1	1.05	2.15	0.12
Nominal Speed (rpm)	1377	1858	1053	156
Stroke short	68	–	–	–
Nominal Torque (Nm)	1.05	–	–	–
Nominal Speed (rpm)	1377	–	–	–

For the rpm values, it was necessary to perform a computation considering the hardware architecture of the motion axes. The values displayed in the traces for the velocity are derivatives calculation computed run-time by the software on the base of the actual position of the axis. The position is referred to the actual position of the motor that is performing the action on the packages. Then, these parameters must be reported to the actual values of the actuator which is the input of the gearbox. To do so the following operation was used:

$$RPM = \frac{[grad/s]}{[grad/rev]} * 60 * \frac{GOut}{GIn} \quad (2.1)$$

Thanks to this, it was possible to obtain the actual velocities of the brushless. To consider the maximum velocity reached by the actuators, it is considered the maximum peak of the axes velocity trace and then converted to the motor speed accordingly to the equation above.

For the torques, it is necessary to analyse the currents. In the traces, these are the actual currents circulating in the motors making it only necessary to apply to each maximum current reached during the cycle the Constant Torque (Nm/A) of the motor itself. At this point, the maximum values of torque and speed are found. To obtain the nominal values of them it is necessary to consider the rms of both the speed and the torque. Unfortunately, Schneider's software does not allow to obtain such values of a single part of the trace. Therefore, I performed a manual computation of these parameters by approximating in appropriated small intervals of the traces the values of speed and current for each axis in both the operative condition of the machine tested. As a result:

	Axis speed	Nominal Torque	Estimation at 100%
Layer	55%	0,844	1,535
	75%	1,099	1,465
Pusher	24 u/min	0,33	0,824
	60 u/min	1.05	–
Crank	24 u/min	0,779	1,947
	60 u/min	2.149	–
Film Pull	24 u/min	0,12	0,298
	60 u/min	0,115	–

At this point, the considerations on the performances of the Schneider actuators are completed. These are now used in order to perform the analysis of the possible Siemens alternatives in the next section.



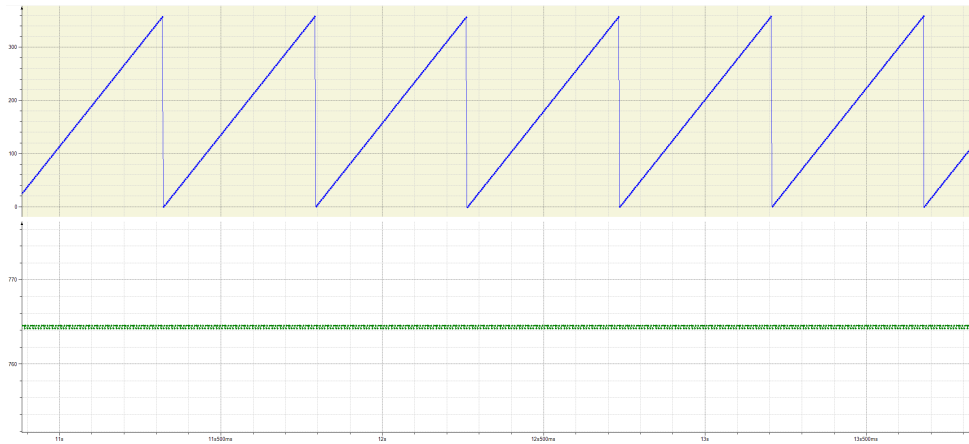


Figure 2.10: Master of the layer axis at 55% speed in no load run.

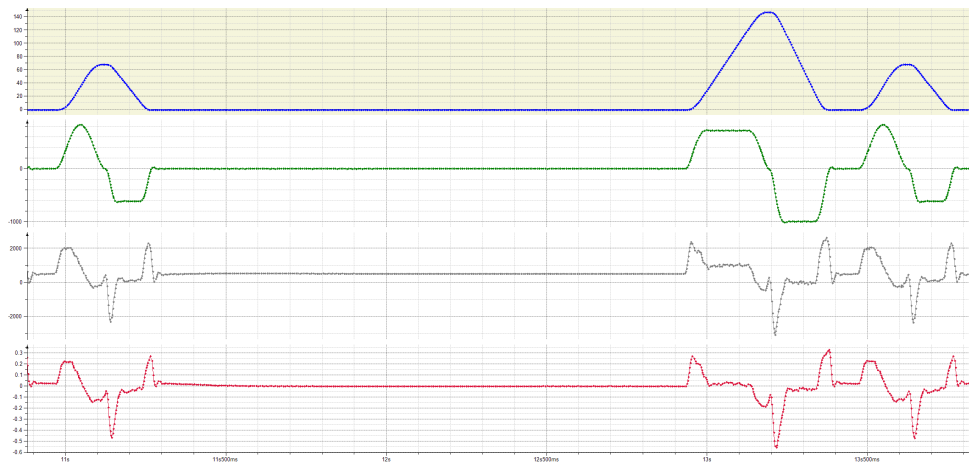


Figure 2.11: Layer axis at 55% speed in no load run.

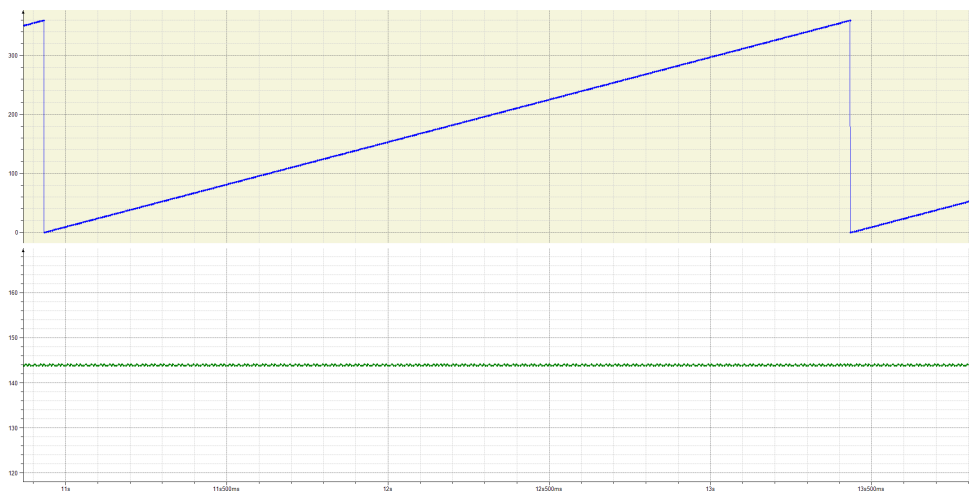


Figure 2.12: Master of the machine axes at 24 u/min in no load run.

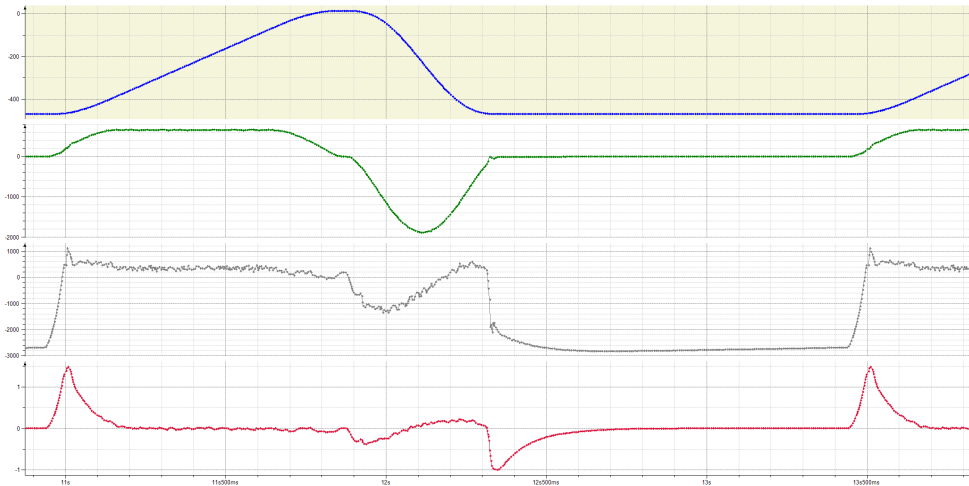


Figure 2.13: Pusher axis 24 u/min in no load run.



Figure 2.14: Film Pull axis 24 u/min in no load run.

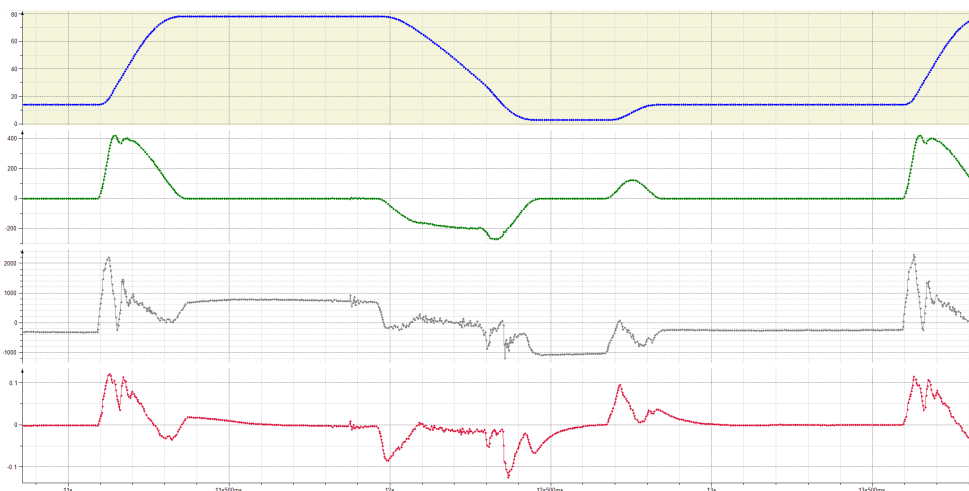


Figure 2.15: Crank axis 24 u/min in no load run.

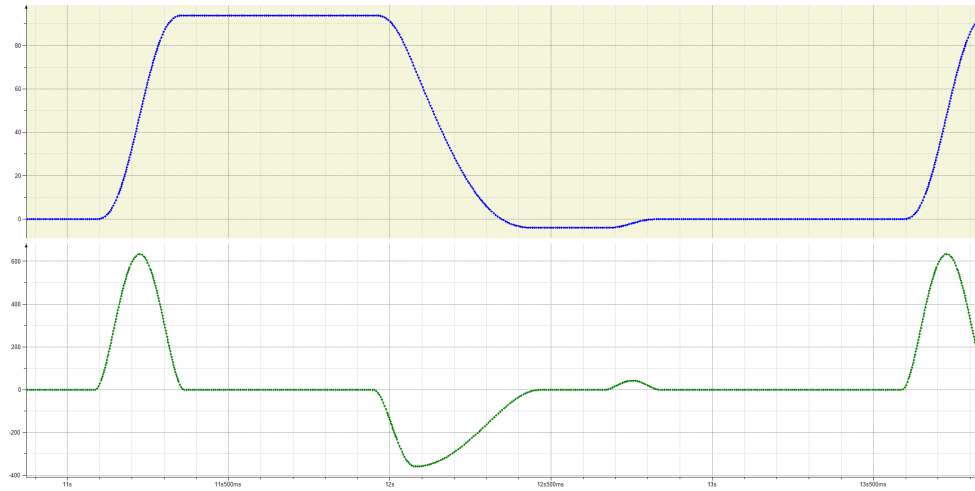


Figure 2.16: Virtual Crank axis at 24 u/min in no load run.

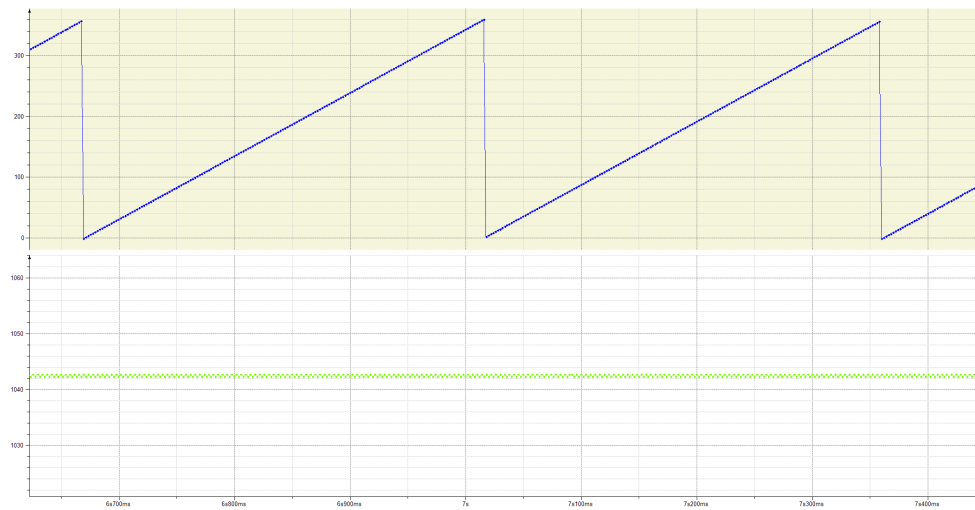


Figure 2.17: Master of the layer axis at 75% speed in no load run.

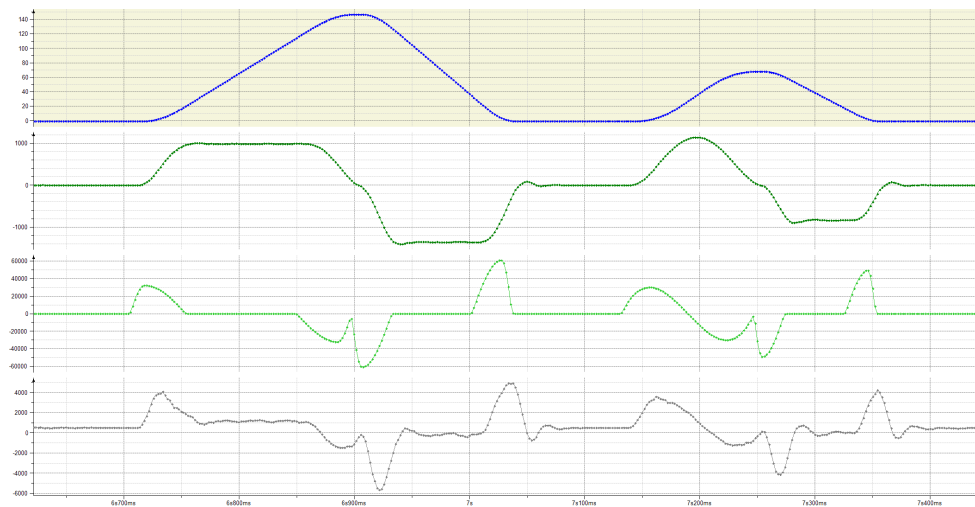


Figure 2.18: Layer axis 60 u/min in no load run.

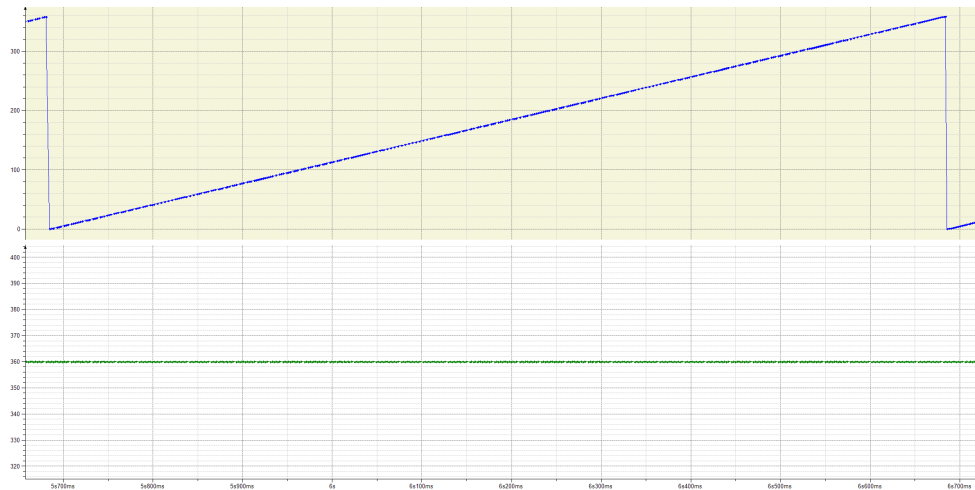


Figure 2.19: Master of the machine axes at 60 u/min in no load run.

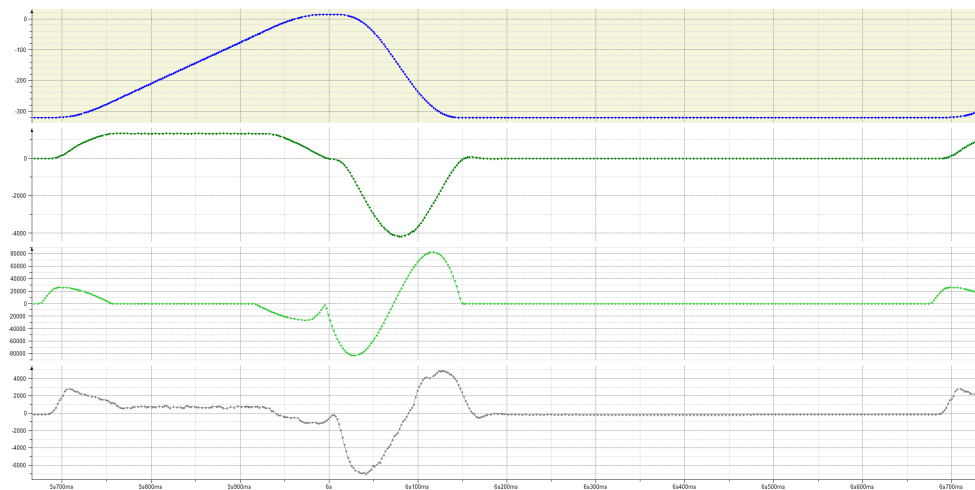


Figure 2.20: Pusher axis 60 u/min in no load run.

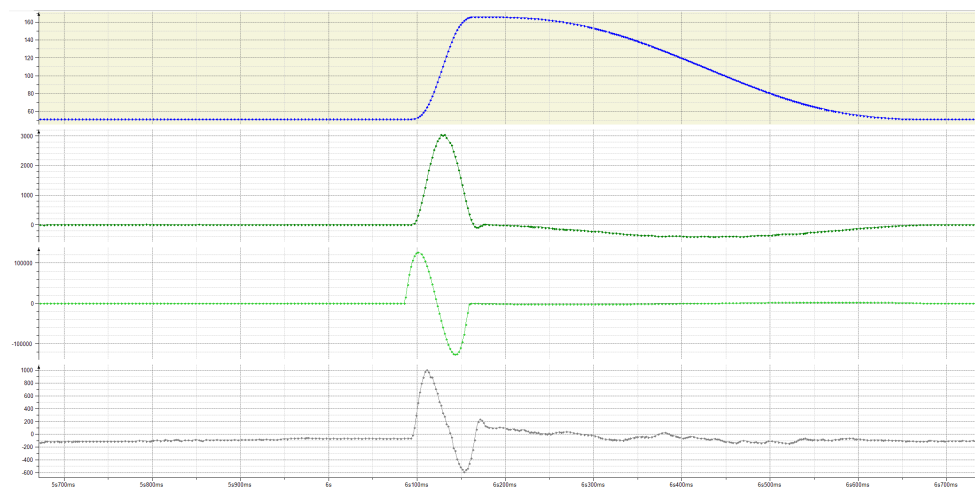


Figure 2.21: Film Pull axis 60 u/min in no load run.

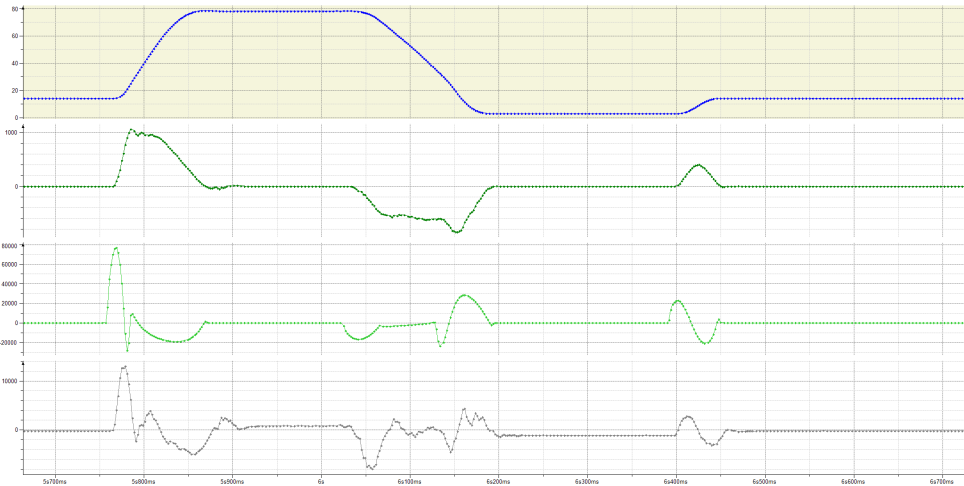


Figure 2.22: Crank axis 60 u/min in no load run.

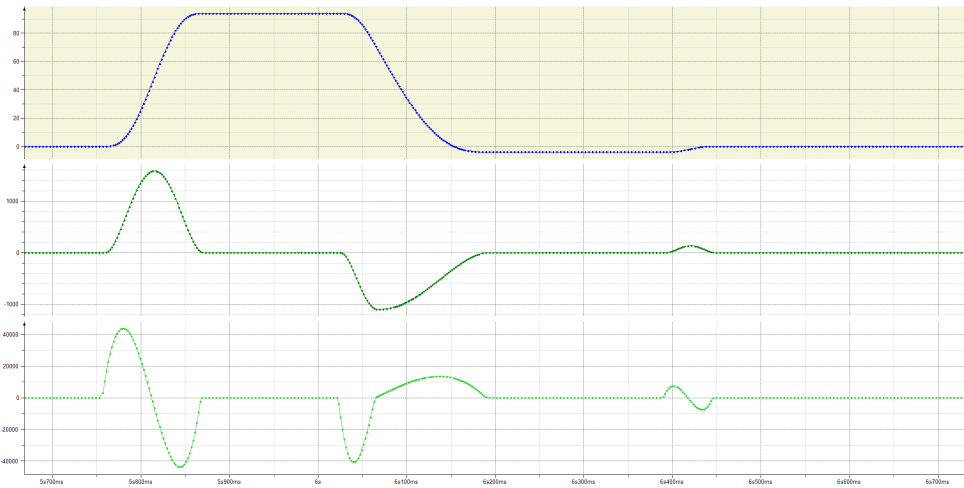


Figure 2.23: Virtual Crank axis at 60 u/min in no load run.

On the base of these information, now how the new motors were chosen is presented. Two different approaches are used. The first one is related to maintain the same motors sizes already mounted on the MS260. This is in line with industry policy to limit component diversity in the magazine. For this reason, the results obtained from the tests show a low level of use of the capabilities of the motor. This is mostly the case of the Film Pull actuator. The values of speed and torque reached are very far from the nominal and, in particular, the maximum values. To limit the need for a larger warehouse, some actuators are oversized compared to real needs. Therefore, the Schneider motors characteristics are used in order to find compatible actuators in the Siemens catalogue. They are added in the DT Configurator which, as mentioned, allows to find the motor when the properties are already known. For the SH3070, the main data can be taken from its data-sheet displayed in Fig. 2.24 on the next page. From these, the result obtained is identified with the motors 1FK7034-2AK71 and 1FT7034-5AK71. The same procedure can be considered also for the Crank

## Presentazione

Compatibilità gamma	PacDrive 3
Tipo di prodotto o componente	Servo motore
Nome dispositivo	SH3.....P

## Caratteristiche tecniche

Massima velocità meccanica	6000 Rpm
Tensione alimentazione nominale [Us]	115...480 V
Numero di fasi della rete	Trifase
Corrente di stallo continua	4,8 A
Coppia di stallo continua	5,8 Nm a 115...480 V trifase
Potenza continua	1900 W
Coppia di stallo max (picco)	18,3 Nm a 115...480 V trifase
Potenza nominale di uscita	580 W a 115 V trifase 1090 W a 230 V trifase 1930 W a 400 V trifase 2210 W a 480 V trifase
Coppia nominale	5,5 Nm a 115 V trifase 5,2 Nm a 230 V trifase 4,6 Nm a 400 V trifase 4,4 Nm a 480 V trifase
Nominal speed	1000 rpm a 115 V trifase 2000 rpm a 230 V trifase 4000 rpm a 400 V trifase 4800 giri/min a 480 V trifase
Irms corrente max	17,1 A
Tipo di albero	Con chiavetta
Diametro dell'albero	19 Mm
Lunghezza albero	40 Mm
Larghezza chiave	6 Mm
Grado di protezione IP	IP54 albero senza anello di tenuta: conforme a EN/IEC 60034-5 IP65 motore: conforme a EN/IEC 60034-5 IP65 bronzina dell'albero: conforme a EN/IEC 60034-5
Tipo di encoder	SinCos Hiperface assoluto monogiro
Risoluzione del segnale velocità	128 Sin/Cos per rivoluzione
Freno di stazionamento	Con
Coppia di attesa	9 Nm
Supporto per montaggio	Flangia standard internazionale
Dimensione flangia	100 Mm
Costante coppia	1,21 Nm/A a 120 °C
Back emf constant	77 V/Krpm a 20 °C
Numero di poli motore	4
Inerzia del rotore	2,928 Kg.Cm <sup>2</sup>
Resistenza statore	2,4 Ohm
Induttanza statore	19,5 MH
Forza radiale max Fr	990 N a 1000 rpm 790 N a 2000 rpm 690 N a 3000 rpm 620 N a 4000 rpm
Forza assiale max Fa	160 N
Tipo di raffreddamento	Convezione naturale
Lunghezza	234,5 Mm
Diametro collare di centraggio	95 Mm
Profondità collare di centraggio	3,5 Mm
Numero di fori di montaggio	4
Diametro dei fori di montaggio	8,5 Mm
Diametro del cerchio dei fori di montaggio	115 Mm
Peso prodotto	6,3 Kg

Figure 2.24: Data sheet of 3070 Schneider actuator.

## Presentazione

Compatibilità gamma	PacDrive 3
Tipo di prodotto o componente	Servo motore
Nome dispositivo	SH3.....P

## Caratteristiche tecniche

Massima velocità meccanica	8000 Rpm
Tensione alimentazione nominale [Us]	115...480 V
Numero di fasi della rete	Trifase
Corrente di stallo continua	2,9 A
Coppia di stallo continua	2,2 Nm a 115...480 V trifase
Potenza continua	1130 W
Coppia di stallo max (picco)	7,6 Nm a 115...480 V trifase
Potenza nominale di uscita	340 W a 115 V trifase 660 W a 230 V trifase 1190 W a 400 V trifase 1360 W a 480 V trifase
Coppia nominale	2,15 Nm a 115 V trifase 2,1 Nm a 230 V trifase 1,9 Nm a 400 V trifase 1,8 Nm a 480 V trifase
Nominal speed	1500 rpm a 115 V trifase 3000 rpm a 230 V trifase 6000 rpm a 400 V trifase 7200 rpm a 480 V trifase
Irms corrente max	11,8 A
Tipo di albero	Con chiavetta
Diametro dell'albero	11 Mm
Lunghezza albero	23 Mm
Larghezza chiave	4 Mm
Grado di protezione IP	IP54 albero senza anello di tenuta: conforme a EN/IEC 60034-5 IP65 motore: conforme a EN/IEC 60034-5 IP65 bronzina dell'albero: conforme a EN/IEC 60034-5
Tipo di encoder	SinCos Hiperface assoluto monogiro
Risoluzione del segnale velocità	128 Sin/Cos per rivoluzione
Freno di stazionamento	Senza
Supporto per montaggio	Flangia standard internazionale
Dimensione flangia	70 Mm
Costante coppia	0,77 Nm/A a 120 °C
Back emf constant	48 V/Krpm a 20 °C
Numero di poli motore	3
Inerzia del rotore	0,41 Kg.Cm <sup>2</sup>
Resistenza statore	4,2 Ohm
Induttanza statore	29,65 MH
Forza radiale max Fr	710 N a 1000 rpm 560 N a 2000 rpm 490 N a 3000 rpm 450 N a 4000 rpm 410 N a 5000 rpm 390 N a 6000 rpm
Forza assiale max Fa	80 N
Tipo di raffreddamento	Convezione naturale
Lunghezza	187 Mm
Diametro collare di centraggio	60 Mm
Profondità collare di centraggio	2,5 Mm
Numero di fori di montaggio	4
Diametro dei fori di montaggio	5,5 Mm
Diametro del cerchio dei fori di montaggio	82 Mm
Peso prodotto	2,8 Kg

Figure 2.25: Data sheet of 3100 Schneider actuator.

axis motor, SH3100. The data taken for the data-sheet in Fig. 2.25 on the preceding page allow finding the correspondent motor in the 1FK7061-4CH71. The second approach, on the other hand, consists in starting from the data collected during the tests and identifying the best actuators that meet the parameters obtained. Of course, the data considered are relative to the most stressful operative condition of the machine which is related to the 60 u/min and the 75% speed.

The data collected in the tables above are meaningful only to understand the limits of the actuators in the different conditions. Indeed, they can not be inserted directly in the SIZER as they do not happen at the same time. For this reason, below is explained how the choice was made. To start, the Layering axis is analysed. From Fig.s above, it is possible to see that the more important cycle is the longest one. Indeed, during that the higher speeds are reached (1400 units/sec during the returning phase which coincides with 6300 rpm of the motor) and the higher torques (almost 6 A of current when the motion is inverted). These data are fundamental for the choice of the alternative. They can be used to identify the operative points that are inserted in the SIZER. Moreover, the nominal values of the torque are important. In the case of the tests, the Layer axis was set at speed of 55% and 75%. Therefore, the results of the nominal torques in both cases are not entirely correct. Considering the proportion between them, a nominal torque value at 100% functionality of 1.5 Nm has been estimated.

	Point @MAX_Torque	Point @MAX_Speed
Speed (units/sec)	764	1375
Speed (RPM)	1600	2887
Current (A)	5.7	1.2
Torque (Nm)	4.39	0.924

Adding the data to the configurator, it is possible to see from Fig. 2.26 on the next page that they are displayed in the Torque-Speed plane. At this point the tool allows you to apply a filter search to find compatible actuators, such as sorting by size, nominal speed or torque. The first two motors presented are similar in performances. The 1FK7040-2AK7 and the 1FK7034-2AK7, both belonging to the family of the 1FK7 actuators, despite the different shaft dimension (48mm in the first case and 36mm in the second), offer almost the same characteristics. By the way, as the table of data and the Figures show, the first one gives a little less maximum torque. Considering the curves with respect to the operative conditions added, both have almost overlapped nominal values with the 100% speed condition of the motor. However, these conditions exit the nominal values only for the higher speed where, as the point at the maximum speed of the 75% condition shows, it is lower than the expected nominal torque. For this reason, both motors are a good alternative. Instead, for the third motor presented in the table and in Fig. 2.29 on page 58, the characteristics are slightly oversized with respect to the previous two. Both the nominal and the maximum torques are larger than before and, for the job that the Layer axis needs to perform, this motor is excessively powerful. To



conclude, this is the first alternative presented by the SIZER after the previous two in terms of torques. This means that all the other possible solutions are even bigger in those terms. So, the only possible solutions are the first two.

	P-nom	M-nom	n-nom	M-max	n-max
1FK7040-2AK7	0,69 kW	1,10 Nm	6000 rpm	5,10 Nm	9000 rpm
1FK7034-2AK7	0,63 kW	1,00 Nm	6000 rpm	6,50 Nm	9000 rpm
1FK2204-5AF0	0,75 kW	2,40 Nm	3000 rpm	7,50 Nm	7500 rpm

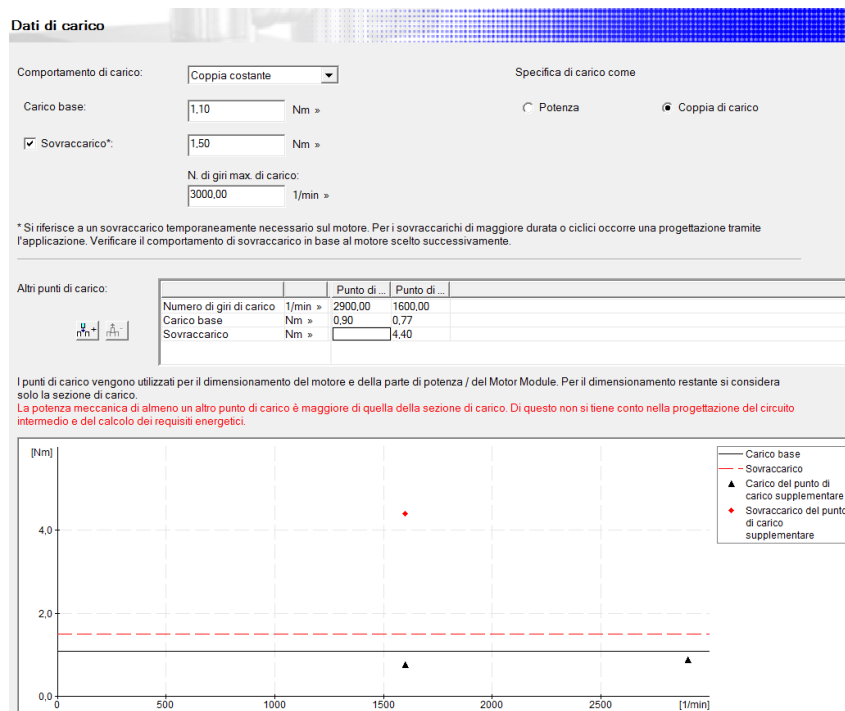
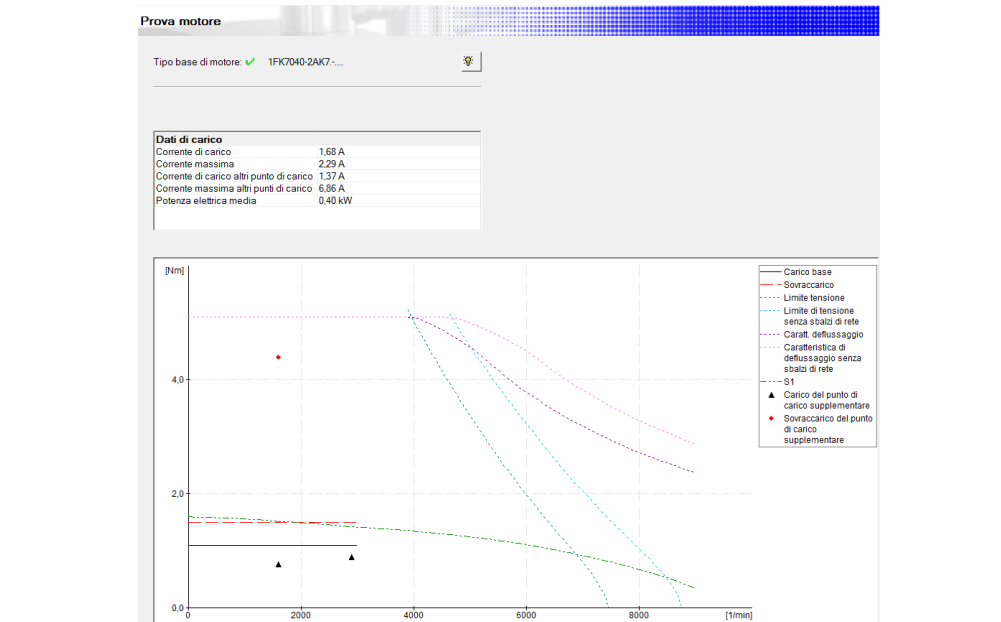


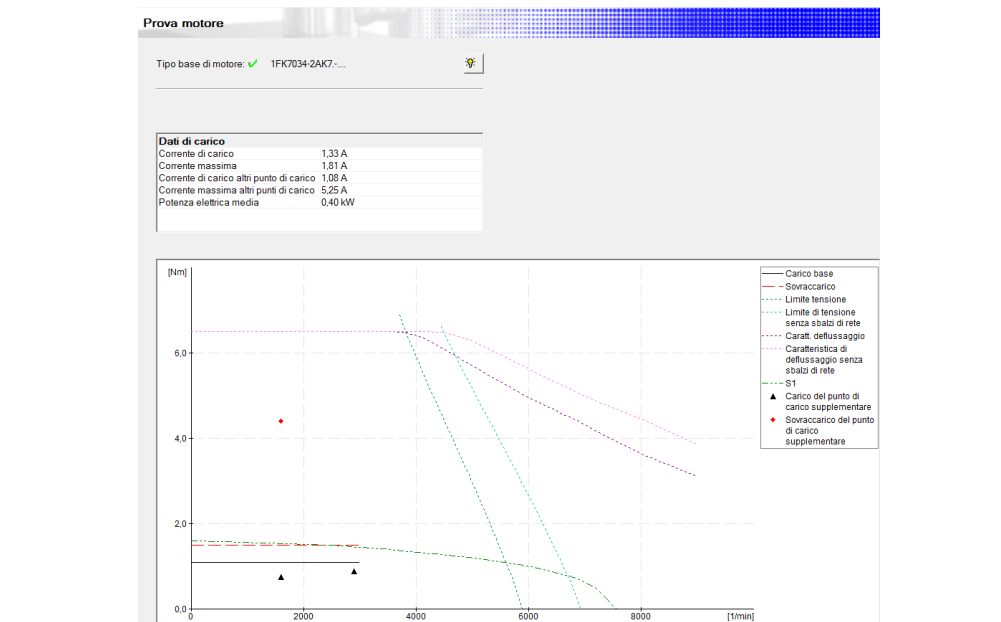
Figure 2.26: Layer Operative Points.



**Criteri di dimensionamento**

Criteri di dimensionamento			
Argomento	Condizione	Valore reale	Valore limite
✓ Numero di giri	Il numero di giri max. del carico è ≤ al numero di giri max. del motore.	3000,00 1/min	9000,00 1/min
✓ Coppia massima	Tutte le coppie di carico sono inferiori alla curva limite della tensione o alla coppia massima del motore, tenendo conto dello sfruttam...	86,3 %	100,0 %
✓ Coppia di carico	Le coppie di carico efficaci (o il carico permanente) sono sotto la curva di carico permanente del motore in considerazione dello sfrutta...	77,5 %	100,0 %

Figure 2.27: Layer 1FK7040-2AK7.



**Criteri di dimensionamento**

Criteri di dimensionamento			
Argomento	Condizione	Valore reale	Valore limite
✓ Numero di giri	Il numero di giri max. del carico è ≤ al numero di giri max. del motore.	3000,00 1/min	9000,00 1/min
✓ Coppia massima	Tutte le coppie di carico sono inferiori alla curva limite della tensione o alla coppia massima del motore, tenendo conto dello sfrutta...	67,7 %	100,0 %
✓ Coppia di carico	Le coppie di carico efficaci (o il carico permanente) sono sotto la curva di carico permanente del motore in considerazione dello sfrutta...	75,7 %	100,0 %

Figure 2.28: Layer 1FK7034-2AK7.

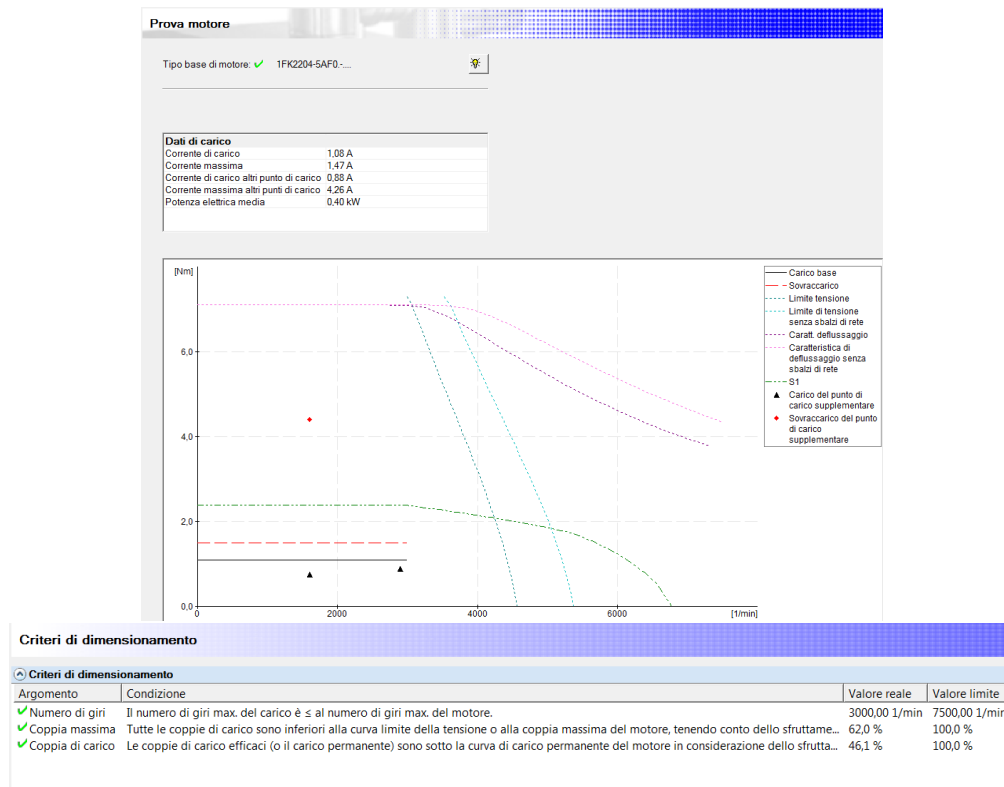


Figure 2.29: Layer 1FK2204-5AF0.

The second axis discussed is the Pusher. As it is also for the Crank and the Film Pull, they are not affected by a parameter that sets the speed, but instead they are affected only by the number of bundles the machine must produce. As a result, among the initial tests on the performances of the motor, the interesting ones are the conditions at 60 u/min for these axes. Therefore, by evaluating more in detail the traces of the Pusher, the operative points requested for the actuator selection can be found. In particular, the table below shows them. It is possible to see how they are different with respect to the Layer axis. In both the points, the maximum value considered is higher in this case than in the previous. Indeed, the torque is 5.4 Nm compared to 4.39 and the speed is 6300 rpm compared to 2887 rpm. Looking at these data it is possible to suppose that the motor required is larger than before. However, considering also the nominal torque reached by the axis during the 60 u/min, it is possible to see from the table below that it is slightly lower. This is due to the fact that the cycle of the second part of the machine controlled by the MCV\_Master is longer than the cycle of the Master\_Layer. As a result, when the data are inserted in the SIZER ( 2.30 on the next page) the motors compatible are the same as before except for one. The 1FK7034-5AK7 motor covers very well the operating conditions having a wide range of speeds covered together with a not too high torque range. Instead, the 1FK2204-6AF0 allows as before grater levels of torques at the expense of the speed. The nominal values are really close to the ones reached by operative conditions leaving a small margin, while the levels of torques are really far from those reached.

	Point @MAX_Torque	Point @MAX_Speed
Speed (units/sec)	2300	4200
Speed (RPM)	3450	6300
Current (A)	7	1.4
Torque (Nm)	5,4	1,08

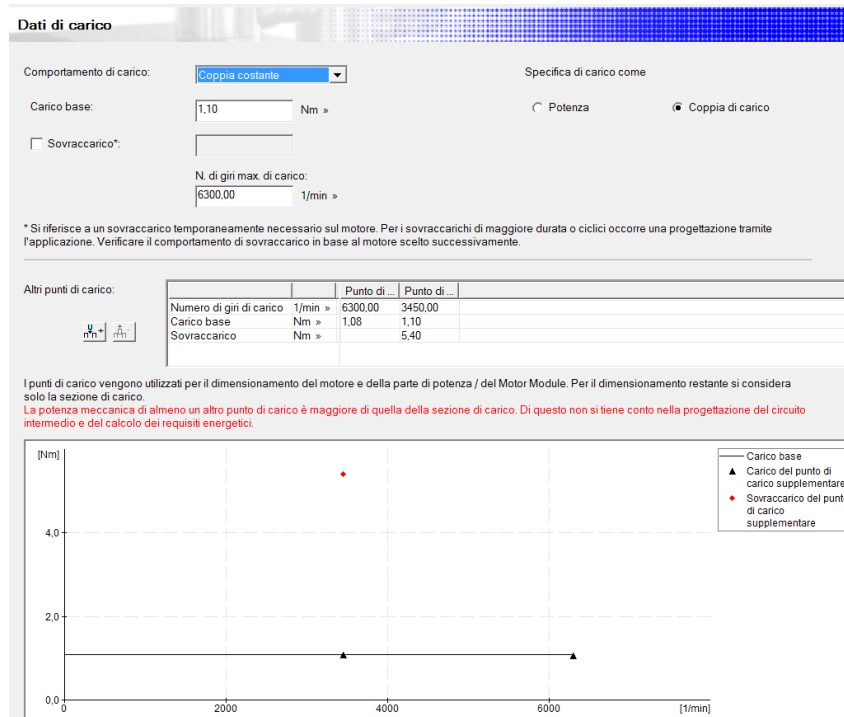
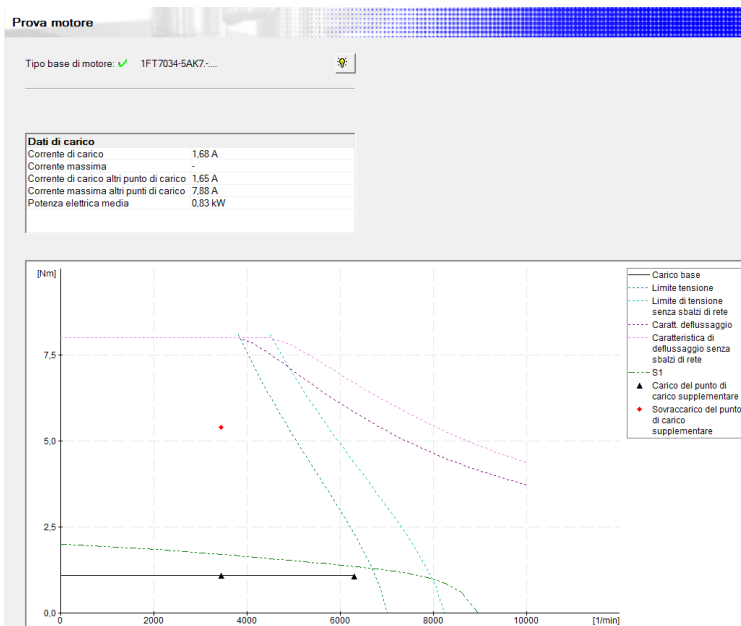


Figure 2.30: Pusher Operative Points.

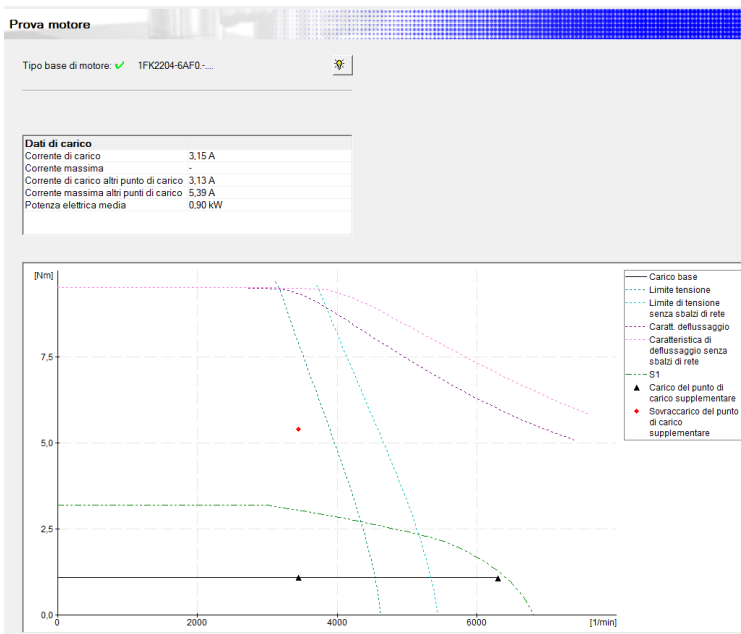


**Criteri di dimensionamento**

⊖ Criteri di dimensionamento

Argomento	Condizione	Valore reale	Valore limite
✓ Numero di giri	Il numero di giri max. del carico è s al numero di giri max. del motore.	6300,00 1/min	10000,00 1/min
✓ Coppia massima	Tutte le coppie di carico sono inferiori alla curva limite della tensione o alla coppia massima del motore, tenendo conto dello sfruttam...	67,5 %	100,0 %
✓ Coppia di carico	Le coppie di carico efficaci (o il carico permanente) sono sotto la curva di carico permanente del motore in considerazione dello sfrut...	80,9 %	100,0 %

Figure 2.31: Pusher 1FK7034-5AK7.



**Criteri di dimensionamento**

⊖ Criteri di dimensionamento

Argomento	Condizione	Valore reale	Valore limite
✓ Numero di giri	Il numero di giri max. del carico è s al numero di giri max. del motore.	6300,00 1/min	7600,00 1/min
✓ Coppia massima	Tutte le coppie di carico sono inferiori alla curva limite della tensione o alla coppia massima del motore, tenendo conto dello sfruttam...	57,9 %	100,0 %
✓ Coppia di carico	Le coppie di carico efficaci (o il carico permanente) sono sotto la curva di carico permanente del motore in considerazione dello sfrut...	85,6 %	100,0 %

Figure 2.32: Pusher 1FK2204-6AF0.

For the third axis considered, that is the Crank, a larger motor was mounted on the machine. The data collected from the traces are referred to as the output shaft of the gearbox. This is important to be underlined because, as is mentioned in the first chapter, the sealing procedure is performed with two axes, a virtual and a real one. The virtual is used in order to follow the cam representing the linear motion that the mechanics should do. These results in position, velocity and acceleration are dependant on the Crank system moved by the motor. Moreover, the virtual axis traces show only values of position, velocity and acceleration and the current is not present. For this reason, it is considered the real axis. From it, the operative points are obtained. As the table below shows, they are characterized by really high values of torque in both cases. This is the consequence of the system attached to it which is characterized by a two-stage gearbox and a crank.

	Point @MAX_Torque	Point @MAX_Speed
Speed (units/sec)	780	1068
Speed (RPM)	2600	3560
Current (A)	13	6,2
Torque (Nm)	15,7	7,5

From the data many motors are compatible. In particular, starting from the plots it is possible to see that two of them are perfectly compatible without reaching the limit of tension area (1FK7060-2AH7 and 1FK7064-4CF7). Instead, in the case of the third compatible motor (1FK7060-2AF7), only the voltage limit without mains surges is not reached. Anyway, this does not affect the compatibility of the motor. The last one, the 1FK7061-4CH7, is the one that was suggested on the base of the Schneider motor but, as the graph 2.34 on the following page shows, the maximum torque requested during the 60 u/min is not possible without exceeding the field of weakening. For the compatible motors, a further comparison can be carried out. Indeed, the 1FK7060-2AH7 and the 1FK7060-2AF7 are similar motors in terms of maximum torque and speed. For the 1FK7064-4CF7, instead, the maximum torque is higher while the speed is lower. This does not give any possible range of exploiting better the performances of the motor in case a higher speed of the machine is requested. On the contrary, the other two motors could allow an increased performance as a higher operating speed of the motor that would bring higher values of torques can be covered up. Especially, this is the case of the 1FK7060-2AH7 motor in which the limit area has not yet been reached. To conclude, these are only the firsts suggested by the SIZER filtered with respect to the torque.

	P-nom	M-nom	n-nom	M-max	n-max
1FK7015-5AK7	0,10 kW	0,16 Nm	6000 rpm	1,00 Nm	8000 rpm
1FK7022-5AK7	0,38 kW	0,6 Nm	6000 rpm	3,40 Nm	9000 rpm

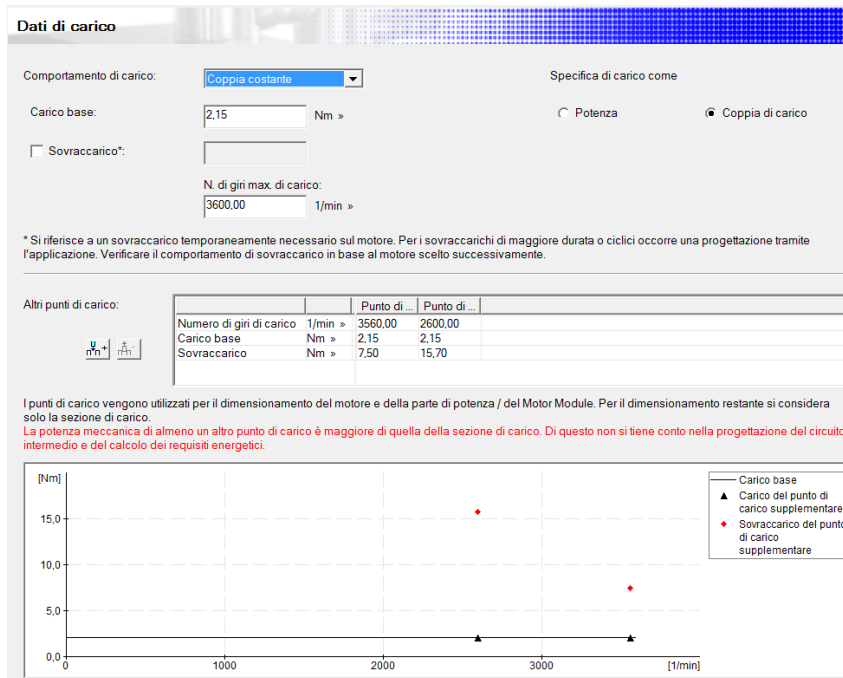
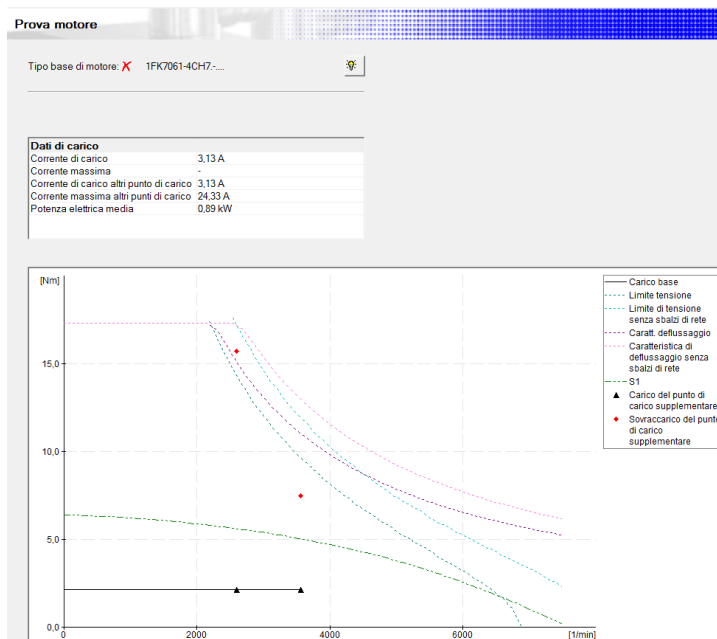


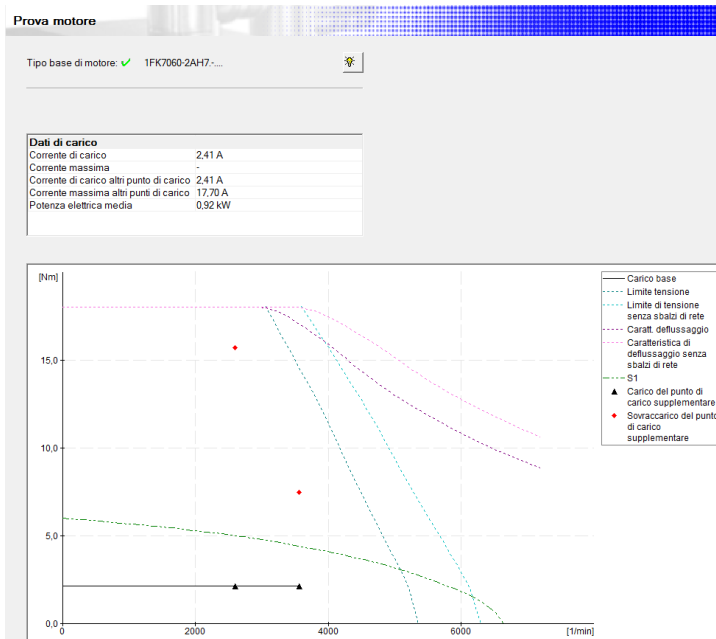
Figure 2.33: Crank Operative Points.



**Criteria di dimensionamento**

Argomento	Condizione	Valore reale	Valore limite
✓ Numero di giri	Il numero di giri max. del carico è ≤ al numero di giri max. del motore.	3600,00 1/min	7500,00 1/min
✗ Coppia massima	Tutte le coppie di carico sono inferiori alla curva limite della tensione o alla coppia massima del motore, tenendo conto dello sfruttam...	104,0 %	100,0 %
✓ Coppia di carico	Le coppie di carico efficaci (o il carico permanente) sono sotto la curva di carico permanente del motore in considerazione dello sfrutta...	42,8 %	100,0 %

Figure 2.34: Crank 1FK7061-4CH7.

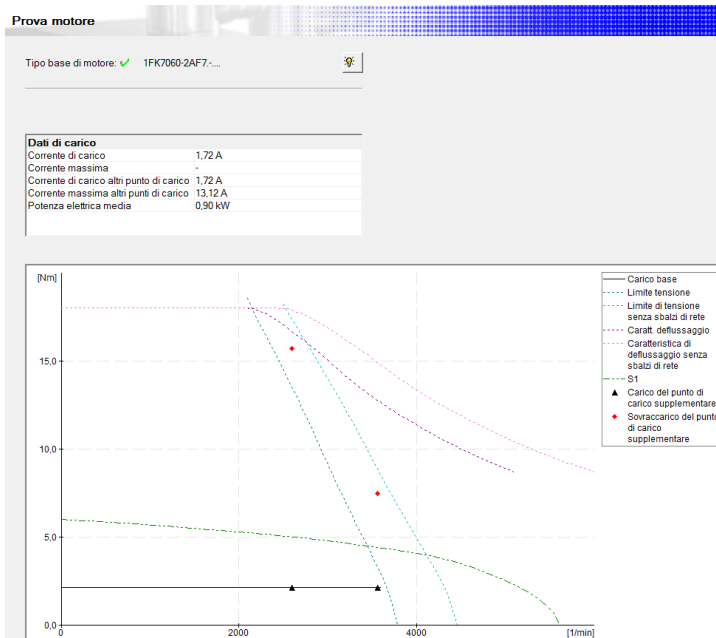


**Criteri di dimensionamento**

○ Criteri di dimensionamento

Argomento	Condizione	Valore reale	Valore limite
✓ Numero di giri	Il numero di giri max. del carico è ≤ al numero di giri max. del motore.	3600,00 1/min	7200,00 1/min
✓ Coppia massima	Tutte le coppie di carico sono inferiori alla curva limite della tensione o alla coppia massima del motore, tenendo conto dello sfruttam...	87,2 %	100,0 %
✓ Coppia di carico	Le coppie di carico efficaci (o il carico permanente) sono sotto la curva di carico permanente del motore in considerazione dello sfrutta...	48,9 %	100,0 %

Figure 2.35: Crank 1FK7060-2AH7.



**Criteri di dimensionamento**

○ Criteri di dimensionamento

Argomento	Condizione	Valore reale	Valore limite
✓ Numero di giri	Il numero di giri max. del carico è ≤ al numero di giri max. del motore.	3600,00 1/min	6000,00 1/min
✓ Coppia massima	Tutte le coppie di carico sono inferiori alla curva limite della tensione o alla coppia massima del motore, tenendo conto dello sfruttam...	94,2 %	100,0 %
✓ Coppia di carico	Le coppie di carico efficaci (o il carico permanente) sono sotto la curva di carico permanente del motore in considerazione dello sfrutta...	48,9 %	100,0 %

Figure 2.36: Crank 1FK7060-2AF7.



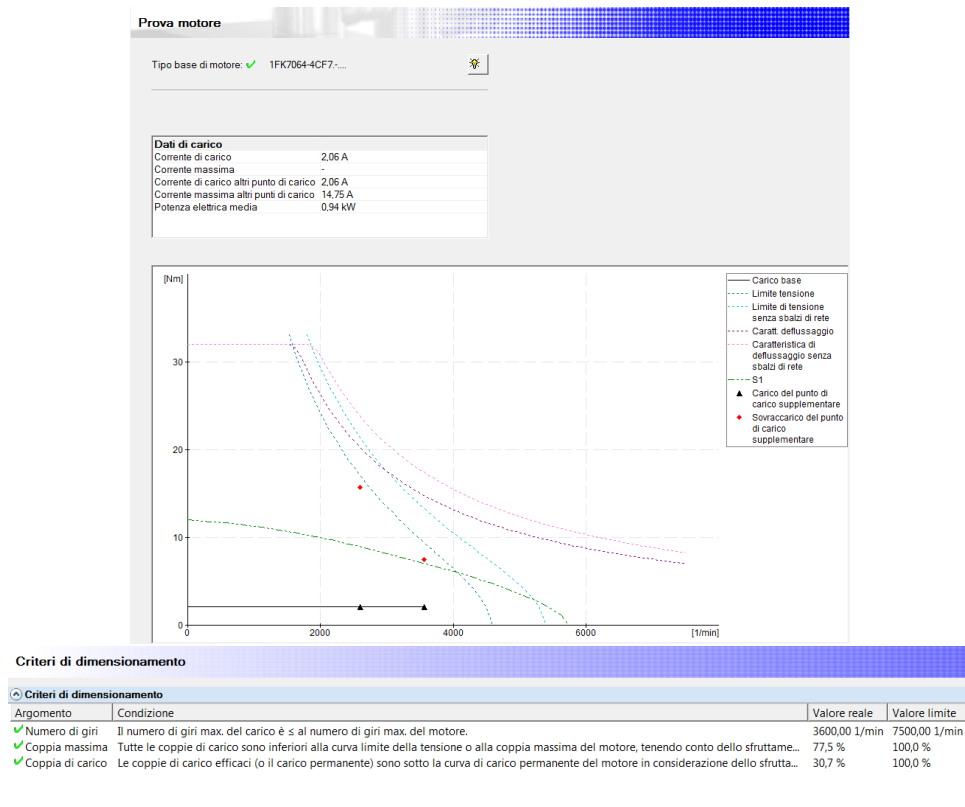


Figure 2.37: Crank 1FK7064-4CF7.

The last axis to be considered is the Film Pull. It has the lowest values in terms of torques and speeds reached. The job done by it is very light compared to the other axes. Moreover, the mechanics attached that it needs to move is very small compared to the other cases. As a result, from the traces taken during the tests and reported in the table below it is possible to see how oversized is the SH3070 mounted. With respect to the Layer and the Pusher which reached values of 4,39 Nm and 2887 rpm in the first case and 5,4 Nm and 6300 rpm in the second, this one has the maximum values of 0,74 Nm and 840 rpm. They correspond to the 15% of torque and 29% of the speed in the worst case. This allows telling that the actuator, presented in the previous discussions and dimensioned on the base of the SH3070, is oversized.

	Point @MAX_Torque	Point @MAX_Speed
Speed (units/sec)	1830	3020
Speed (RPM)	508	840
Current (A)	960	0.35
Torque (Nm)	0.74	0.27

By adding to the SIZER the operative points computed as shown in the table, the resulting motors validate these data. The two smallest motors available are presented. One is the 1FK7015-5AK7 and the other is the 1FK7022-5AK7. The first gives a good alternative. Indeed, the nominal values available cover the nominal operative points of the axis while the peaks reached during the

work are lower than the maximum allowed. In the second alternative, the higher dimensions of the shaft make the nominal and maximum torque of the motor greater. In this case, the nominal curve is sufficient to cover all the operative points as 2.40 on the next page shows. This makes the motor already oversized and therefore not necessary unless under other circumstances. Indeed, the choice of the motor does not rely only on the operative conditions, but also on the cost of it. In case the prices are the same or close to each other, the physical dimension of the motor and the necessity of different gearboxes can make the difference in the choice of one or the other. Therefore, under these circumstances, the motor 1FK7015-5AK7 should appear better for performances, but in case it is not convenient this choice an even larger motor than the 1FK7022-5AK7 can be made. For example, 1FK7034-5AK7 can be used. Indeed, this is the same case of the Schneider version where the SH3070 is used for all the three smaller axes of the machine.

	P-nom	M-nom	n-nom	M-max	n-max
1FK7015-5AK7	0,10 kW	0,16 Nm	6000 rpm	1,00 Nm	8000 rpm
1FK7022-5AK7	0,38 kW	0,6 Nm	6000 rpm	3,40 Nm	9000 rpm

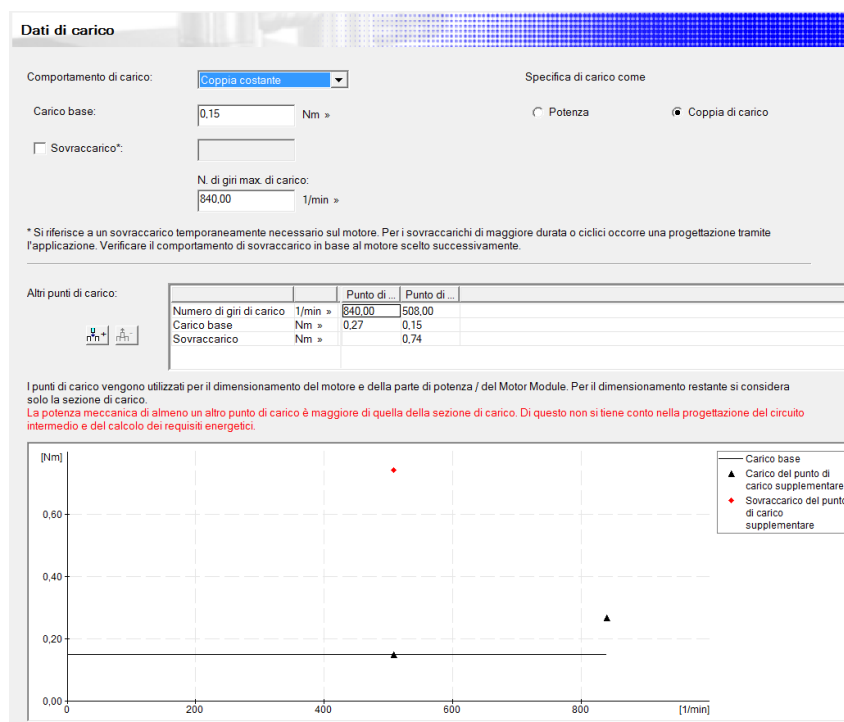
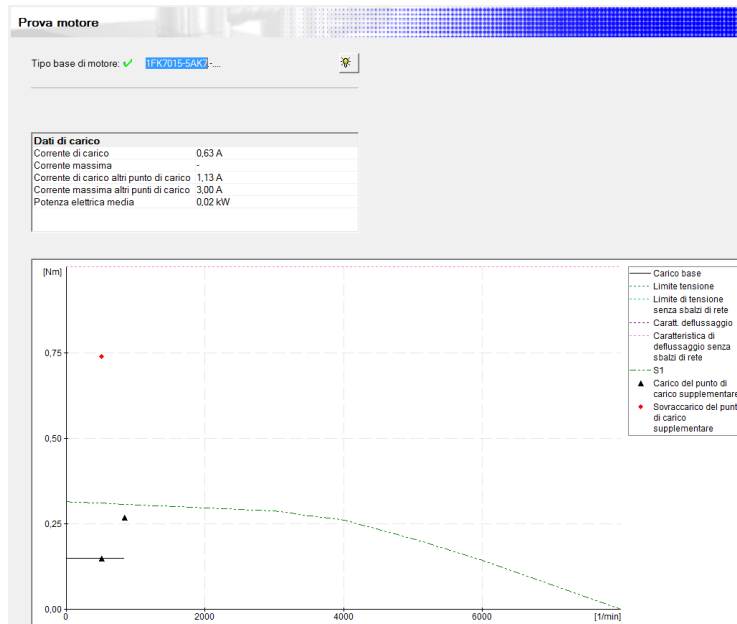


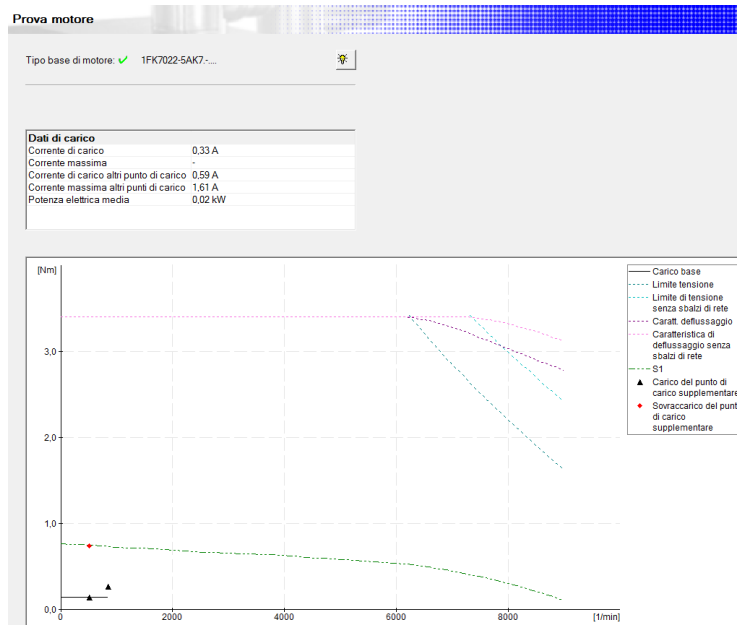
Figure 2.38: Film Pull operative Points.



**Criteri di dimensionamento**

Criteri di dimensionamento			
Argomento	Condizione	Valore reale	Valore limite
✓ Numero di giri	Il numero di giri max. del carico è ≤ al numero di giri max. del motore.	840,00 1/min	8000,00 1/min
✓ Coppia massima	Tutte le coppie di carico sono inferiori alla curva limite della tensione o alla coppia massima del motore, tenendo conto dello sfruttame...	74,0 %	100,0 %
✓ Coppia di carico	Le coppie di carico efficaci (o il carico permanente) sono sotto la curva di carico permanente del motore in considerazione dello sfrutta...	87,8 %	100,0 %

Figure 2.39: 1FK7015-5AK7.



**Criteri di dimensionamento**

Criteri di dimensionamento			
Argomento	Condizione	Valore reale	Valore limite
✓ Numero di giri	Il numero di giri max. del carico è ≤ al numero di giri max. del motore.	840,00 1/min	9000,00 1/min
✓ Coppia massima	Tutte le coppie di carico sono inferiori alla curva limite della tensione o alla coppia massima del motore, tenendo conto dello sfrutta...	21,8 %	100,0 %
✓ Coppia di carico	Le coppie di carico efficaci (o il carico permanente) sono sotto la curva di carico permanente del motore in considerazione dello sfrutta...	36,7 %	100,0 %

Figure 2.40: 1FK7022-5AK7.

# Chapter 3

## Structure of the TIA Portal project

In this chapter, the organization of the solution developed for the new motion control of the MS260 is presented. It covers multiple components of the TIA Portal project and therefore this chapter is divided into different sections where they are analysed one by one. Fig. 1.1 on page 14 shows the whole project tree of the new algorithm. It allows showing the organization of the new motion control before starting to enter in the detail of each part. However, only the PLC folder was considered in the first chapter because it contains the presentation of the previous machine code. Now, in this chapter, all the project tree is analysed. To this porpoise, Fig. 3.1 on page 69 shows the other important parts exploded to better appreciate the different aspects.

### 3.1 Code structure

Before entering the details, the structure of the project tree must be explained. In Fig. 1.1 on page 14, a major division can be done. One of them contains the algorithm. It is organised in different parts each of them with a different key role. The *Program Blocks* folder contains all the programmable blocks used for the control algorithm of the machine. Fig. referenced above shows how it is organized. The first chapter entered the details of the PLC folder which is the only part transferred from the old project control algorithm of the MS260. In addition to it, Fig. allows identifying three additional folders. By looking at Fig. 3.1 on page 69, they are:

- The *Motion* folder is the central topic that is going to be explained in this chapter and analysed in the next one. It contains the new motion control algorithm. It takes advantage of several pre-made libraries from Siemens and, in some specific cases, built ad-hoc for the functionalities not supported by the software.
- The *ReadCam* folder is an additional functionality added to the PLC that allows to transfer and show on the HMI screen the cams active. It

---

is not fundamental for the motion control and it affects mostly the HMI. Therefore, it is discussed in the next chapter.

- The *DB\_SD\_Export\_Import* folder is, likely the previous, an additional functionality. It allows transferring Timers and System data from the PLC to and SD card and the other way around. It is similar to the previous one for the work done in relation to the PLC. Therefore, it is also analysed in the next chapter.

The *Technology objects* contains the representation of real objects in the controller. They are necessary for the execution of the motion control and they are going to be analysed later. The remaining folders of interests are the *PLC data types* and the *PLC Tags*. In the case of the former, it contains the definition of the structures used inside the code and, in the case of the latter, the constants or global variable to be accessed.

The second division contains the physical hardware that is attached to the main controller and then must be added and configured in the project (Drive unit in Fig. 3.1 on the following page). Along with the virtual objects required by the PLC (Technology objects), it represents the interface between the new hardware components and the PLC. It is an important aspect because it prepares to understand how Siemens manages the axes and the drives. Therefore, the two components are presented before the actual motion control algorithm.

The successive sections show the code and its libraries. Entering in the detail of the motion control is the core part of the newly designed control. In Fig., under the *Program blocks* the main programs (in purple) that are called cyclically by the CPU can be seen. Then, the successive folders contain, from the top, the libraries used during the development of the code. The *Motion* folder, instead, contains the main code written for the motion control and it is analysed after the other parts.

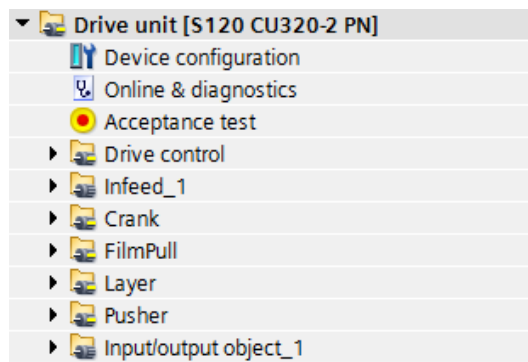
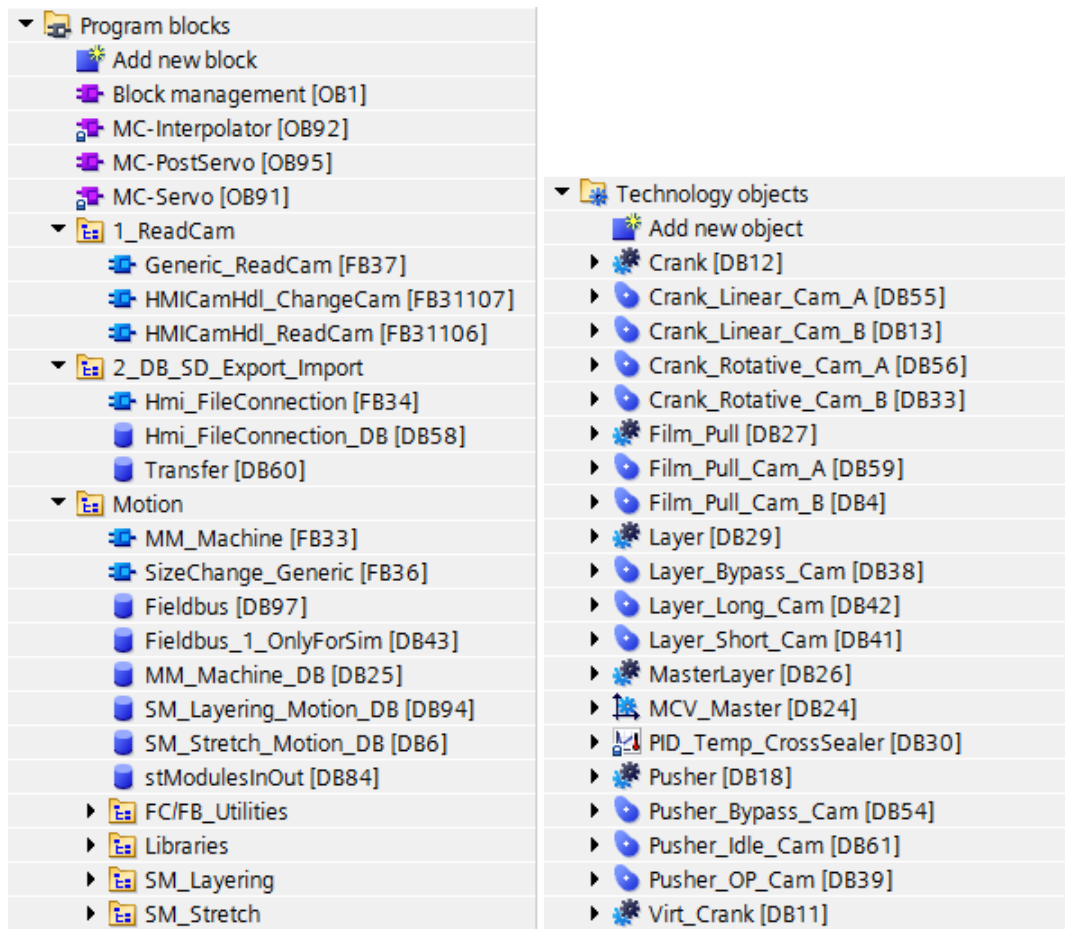


Figure 3.1: Project structure.

---

## 3.2 Technology objects

Technology objects represent real objects (e.g. a drive) in the controller. The functions of the technology objects are called by means of Motion Control instructions in the user program.

The technology objects provide open- and closed-loop control of the movement of the real objects and report status information (e.g. the current position). The configuration of the technology objects represents the properties of the real object. This is stored in a technology data block. The following technology objects are available for Motion Control:

- Speed axis technology object:  
The speed axis technology object (TO\_SpeedAxis) is used to specify the speed for a drive;
- Positioning axis technology object:  
The positioning axis technology object (TO\_PositioningAxis) is used to position a drive with closed-loop position control;
- Synchronous axis technology object:  
The synchronous axis technology object (TO\_SynchronousAxis) includes all functions of the positioning axis technology object. The axis can also be interconnected with a leading value so that the axis follows the position change of a leading axis in synchronous operation;
- External encoder technology object:  
The external encoder technology object (TO\_ExternalEncoder) detects a position and makes it available to the controller;
- Measuring input technology object:  
The measuring input technology object (TO\_MeasuringInput) detects actual positions quickly, accurately and event triggered;
- Output cam technology object:  
The output cam technology object (TO\_OutputCam) generates switching signals depending on the position of an axis or external encoder. You can evaluate the switching signals in the user program or feed them to digital outputs;
- Cam track technology object:  
The cam track technology object (TO\_CamTrack) generates a switching signal sequence depending on the position of an axis or external encoder. In this process, up to 32 individual cams are superimposed and the switching signals are output as a track;
- Cam technology object (S7-1500T):  
The cam technology object ("TO\_Cam") defines a function  $f(x)$  by means of interpolation points and/or segments. Missing function ranges are interpolated;

- 
- Kinematic technology object (S7-1500T):  
The Kinematic technology object (TO\_Kinematics) is used to interconnect positioning axes to a kinematic. This allows to configure the kinematics technology object and to interconnect the axes in accordance with the configured kinematics type.

### 3.2.1 TO\_Axes

Regarding the axes choice, these type of TO must be parameterized when added to the project. This procedure is very simple and intuitive. Opening the "Configuration" page of the newly added axis, different windows can be opened. The ones of interest are mostly: the Basic parameters and the Mechanics. In the first one, the nature of the axis is defined. Specifically, if it is a real or virtual axis is defined. In addition to this, in case the real axis is not actually physically implemented in the mechanical configuration of the system, the "Simulation" setting can be enabled. In this way, it is treated as a virtual axis but the components of the drive that are defined and linked to it inside the project can be left in the project even if they are not present in the machine. In this window, the axis must be also set as a linear or rotating one along with the units of measure that characterize its control. Moreover, the "Modulo" setting is present. When an axis moves in only one direction, the position value continually increases. To limit the position value to a recurring reference system, this setting can be enabled. When the "Modulo" setting is activated, the position of the Technology Object is mapped onto a recurring modulo range. The modulo range is defined by the start value and length. For example, in case of a virtual master, to limit the position value, the modulo range can be defined with start value =  $0^\circ$  and length =  $360^\circ$ . As a result, the position value is mapped onto the modulo range  $0^\circ$  to  $359.999^\circ$ . In the second window, the mechanical components attached to the motors are defined. In particular, the gearbox ratio and the "Leadscrew pitch" are present. The latter is required only in case a linear motion is defined in the "Basic parameters window". Indeed, these last two elements allow the software to calculate the conversion between the motion required to the final axis of the mechanical system and the motion that the actuator must perform to satisfy that request.

Looking at the definition above, the axes of interest are re-conducted to the TO\_PositioningAxis and TO\_SynchronousAxis. These are the only two which allow a positioning and a cam coupling of the axes. In addition to them, in the project are also exploited the TO\_OutputCam through which it is possible to establish the synchronization of the axes with the pneumatic actuators and the check of the operative conditions. Lastly, the recently added objects with the new version of the PLC are exploited. In particular, this is the case of the TO\_Cam which allows the definition of electronic cams paired with the motion of the real axes. Accordingly to the structure of the machine and to the project choices made in the motion control algorithm, the technology objects displayed in Fig. 3.1 on page 69 are used. In particular, as it is explained in the previous chapter, the machine motion control is organized into two



---

different sub-modules (SM\_Layer and SM\_Stretch) which contain respectively one virtual and one real axes and two virtual and three real axes.

In the first case, the axes of reference are the virtual *MasterLayer* and the real *Layer*. Considering the definitions of technology objects, the virtual axis should be defined as a positioning one. Indeed, its main goal is to drive the real axis through a cam coupling. However, in the project, it is built as a synchronous axis. This choice is made on the base of a specific operative condition that is explained successively when the motion algorithm is analysed. Namely, this is a condition where the first sub-module is directly connected to the virtual axis of the second. This requires manipulation of the virtual *MasterLayer* as a synchronous axis. In any case, the alternatives related to this solution are analysed in the next chapter. For the definition of the parameters of this axis, it is sufficient to specify only its virtual nature along with the mechanical structure through which it drives the paired axes. Namely, it is defined as a rotating axis with default dynamic and limit values as, for the virtual axes, the software does not consider them as strict bounds and executes whatever action is commanded him. An important parameter in the definition of a virtual axis is the "modulo" setting that here is set to 360°. For the *Layer* axis, some parameters need instead to be defined. This is required from the mechanical composition of the piston that the actuator needs to move. As a result, it is defined as a linear motor and the units of measure related are in mm, mm/s, Nm and N. For what concerns the driving axis of this synchronous object, it is paired with the *MasterLayer*. Indeed, as mentioned above, this last one is then paired with the *MCV\_Master*. In this way, by transition, it controls the real axis that here is considered. Regarding the remaining mechanical parameters, the gear ratio of the reducer and the belt motion conversion one defined and explained in the previous chapter are also added.

In the second case, the virtual axes present are the *MCV\_Master*, *Virt\_Crank* and the real ones are the *Crank*, *Film\_Pull* and the *Pusher*. The virtual axis *MCV\_Master* is managed as the one presented before. Indeed, it is a rotating axis with modulo 360°. About the mechanical settings, nothing is defined as it is, indeed, a virtual axis. Differently than before, it is built as a *TO\_PositioningAxis* as it is not driven in any circumstances by other axes. As a result, there is no need to make use of PLC resources for a *TO\_SynchronousAxis* that is not exploited. The second virtual axis is analysed with the *Crank* axis. These two are a pair of axes linked to the same mechanical part of the machine, namely the sealer. The real axis is used to control the actuator that moves the crank. The virtual one instead is used only for displaying porpoises. The managing of this complex mechanical system of the machine requires a specific library developed during the creation of the code. Indeed, the TIA Portal software does not cover the necessity to transform the linear motion of one end of a crank system to the rotating motion of the other. For this reason, the library I developed allows creating immediately a rotating cam profile paired to the linear one requested by the machine motion. The specifics of the library are exposed in a different section of this chapter. To

---

clarify the use of the two axes, while the real one follows the rotating cam profile, the virtual axis follows the linear one. However, the virtuality of the latter is a necessity because it is controlled in parallel with the real axis but its motion does not affect any other machine component. The position of the virtual axis is used to display the movement of the linear motion of the sealer. As a result, the virtual axis is defined as a `TO_SynchronousAxis`. Regarding the parameters, it is set as a linear motor without a modulo, while no specifications are present for the mechanical one. In the case of the *Crank*, it is defined as the previous one and linked to the *MCV\_Master*. Differently, it is set a rotating axis with the specified gear-ratio of the gearbox used. No further parameters have to be set yet. The *Pusher* is the second axis for importance in this part of the machine. It is required to push the bundle in the sealing position. Then, for its settings, it is still defined as a `TO_SynchronousAxis` linked to the *MCV\_Master* and with a linear motion. For the mechanical part, the data shown in the second chapter are inserted in the relative spots. Lastly, the *Film\_Pull* follows the same steps as it is for the *Crank*. Namely, a real rotating axis linked to the *MCV\_Master* with a specified ratio related to the gearbox mounted on the machine.

### 3.2.2 TO\_OutputCam

Apart from the axes, other technology objects are implemented as Fig. 3.1 on page 69 shows. The more marginal ones are the `TO_OutputCams` linked to the *MCV\_Master*. The output cam technology object generates switching signals depending on the position of an axis or external encoder. The switching states can be evaluated in the user program and fed to digital outputs. The following output cam types can be used:

- Distance output cam:  
Distance output cams switch on between the start position and end position. Outside this range, the distance output cam is switched off. The switch-on range of distance output cams is basically defined by the start position and end position. When the start position is less than the end position, the switch-on range begins with the start position and ends with the end position. Instead, when the start position is greater than the end position, there are two switch-on ranges:
  - Switch-on range beginning with the start position and ending with the positive range end (e.g. positive software limit switch, end of modulo range)

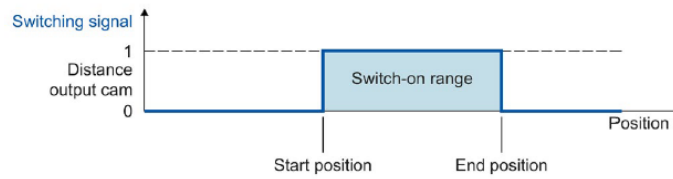


Figure 3.2: TO\_OutputCam Distance-based when start position is lower than the end position.

- Switch-on range beginning with the negative range end (e.g. negative software limit switch, start of modulo range) and ending with the end position

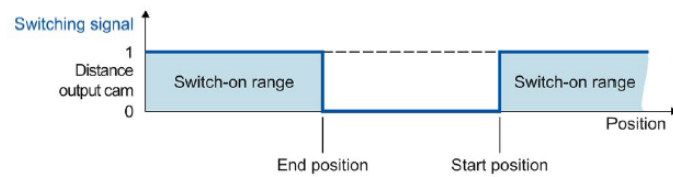


Figure 3.3: TO\_OutputCam Distance-based when start position is greater than the end position.

- Time-based output cam:  
Time-based output cams switch on for a defined time period when the start position is reached.

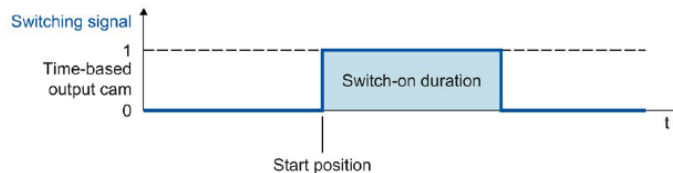


Figure 3.4: TO\_OutputCam Time-based.

An output cam can be switched depending on the motion direction of the interconnected technology object. It is possible to output an output cam with positive or negative motion direction only or even independent of direction. The main difference for this setting between the two types of TO\_OutputCam is that for the time-based one the direction of motion influences the activation according to the direction in which the starting position is reached, while no effects there are on the time duration. In the case of the distance-based one, the activation is affected as the time-base cam but in relation to its distance properties. Moreover, in this case, the cam is deactivated in the case of an inversion in the motion direction of the axis. This type of TO is required to be directly added in the axis of which they look at the position (like Fig. 3.1 on page 69 shows). The details of the different TO\_OutputCam added in the project will be analysed later with the code.

---

### 3.2.3 TO\_Cam

The last TOs used in the project are the cams. They allow establishing the master-slave coupling between an axis and its driving one.

The cam technology object defines a transfer function  $y = f(x)$ . The dependency of an output value on an input value is described in this transfer function in a unit-neutral manner. A cam technology object can be used multiple times. You define the function  $y = f(x)$  in the configuration of the technology object using interpolation points and/or segments. Ranges between interpolation points and segments are interpolated using the Motion Control instruction "MC\_InterpolateCam". The settings can be changed/redefined during runtime of the user program with the Technology DB according to the appendix Tags of the cam technology object.

Commonly, the cam technology object is configured with an editor. The cam is created using a diagram, a table containing the elements of the curve and the properties of the elements. Transitions are calculated between the individual elements of the cam (e.g. points, lines, polynomials). The curve reflects the path-related dependency between the leading axis (leading values, abscissa in the chart) and the following axis (following values, ordinate in the chart). To use a cam in the user program, it must be interpolated after downloading to the CPU. Following the interpolation, the gaps between the defined interpolation points and segments of the cam are closed. The cam is interpolated from the minimum value in the leading value range to the maximum value. To have a faster and easier approach to the cam definition a specific library is available that is going to be presented later.

In the motion control project for the MS260, several cams are used. Their number is related to the number of functions an axis can execute during normal operations. Indeed, the cams can be used multiple times and can even be redefined. Therefore, in order to have a faster approach and to avoid interpolation of the same cam each time a new command is requested, the different operations have different cams. Moreover, as Fig. 3.1 on page 69 shows, the Crank and the FilmPull axes are characterized by an A and a B version of the same cam. This is required because the TIA Portal does not allow the interpolation of the cam while it is in use. Then, to maintain the possibility that the two axes mentioned can have a run-time motion update, a new cam is interpolated and passed to the axis when ready. This alternating process between two cams was not present in the Schneider version. In the case of the Crank, the cams present are the result of the complex motion conversion explained before.

### 3.2.4 Organization blocks

The technology object section of the project is not independent of the programmable part. In Fig. 1.1 on page 14, the structure is composed of particular blocks called OB. These are the blocks executed by the CPU dependently on their nature. A detailed overview is performed in the motion control section presented later in this chapter. To be known for the moment is that, among

---

those present in Fig. previous referenced, three are strictly linked to the TO just described: MC-Servo, MC-Interpolator, MC-PostServo.

- When a technology object is created for S7-1500 Motion Control, the organization blocks "MC-Servo [OB91]" for processing the technology objects are created automatically. The motion control functionality of the technology objects creates its own priority class and is called in the SIMATIC S7-1500 execution system according to the application cycle. The MC-Servo OB is write-protected and the contents cannot be changed.

The position control algorithms of all technology objects configured for motion control on the CPU are calculated within the MC-Servo OB. This means that every time the code needs to control the position of the motors and send the new commands and positions to the drive, this OB is executed. In relation to this, the application cycle and the priority of the organization block in accordance with the project requirements for control quality and system load.

Regarding the application cycle, in which the MC-Servo OB is called, is set in the properties of the organization block:

- Synchronous to the BUS The MC-Servo OB is called synchronously with a bus system. The send clock is editable in the properties of the selected bus system. Two bus systems can be selected: Isochronous PROFIBUS DP, Isochronous PROFINET IO.
- Cyclical The MC-Servo OB is called cyclically with the specified application cycle.

The selected application cycle must be long enough to allow all the technology objects for motion control to be processed in one cycle. If the application cycle is not observed, overflows occur. In this case, the CPU switches to STOP mode.

- When a technology object is created for S7-1500 Motion Control, the organization block "MC-Interpolator [OB91]" for processing the technology objects are created automatically. The motion control functionality of the technology objects creates its own priority class and is called according to the application cycle of the MC-Servo OB in the execution system of the SIMATIC S7-1500. The MC-Interpolator OB is write-protected and the contents cannot be changed. The evaluation of the motion control instructions as well as the monitoring and the set-point generation for all technology objects configured on the CPU (motion control) is performed within the MC-Interpolator OB.

The MC-Interpolator OB is called at the end of the processing of the MC-Servo OB.

When the application cycle of the MC-Servo OB is set, it must be long enough to allow all the technology objects for motion control to be processed in one application cycle. If the application cycle is not observed, overflows occur. Note the following response to overflows: - The CPU tolerates a maximum of three consecutive overflows of the MC-Interpolator OB. -The processing of an MC-Interpolator OB can be interrupted by the calling of the MC Servo OB, at most. If more overflows or interruptions occur, the CPU switches to STOP mode.

- the organization block MC-PostServo [OB95] can be programmed and is called in the application cycle configured for the MC-Servo OB. MC-PostServo [OB95] is called directly after MC-Servo [OB91].

When processing the Motion Control functionality, the organization blocks MC-Servo [OB91] and MC-Interpolator [OB92] are called and processed in each application cycle (processing also occurs in the STOP operating state of the CPU). The remaining cycle time is available for the processing of your user program. For error-free program execution, the following rules must be considered:

- In each application cycle, MC-Servo [OB91] must be started and executed completely.
- In every application cycle, the relevant MC-Interpolator [OB92] must at least be started.

The following figure shows an example of the error-free operational sequence for the processing of organization block OB1:

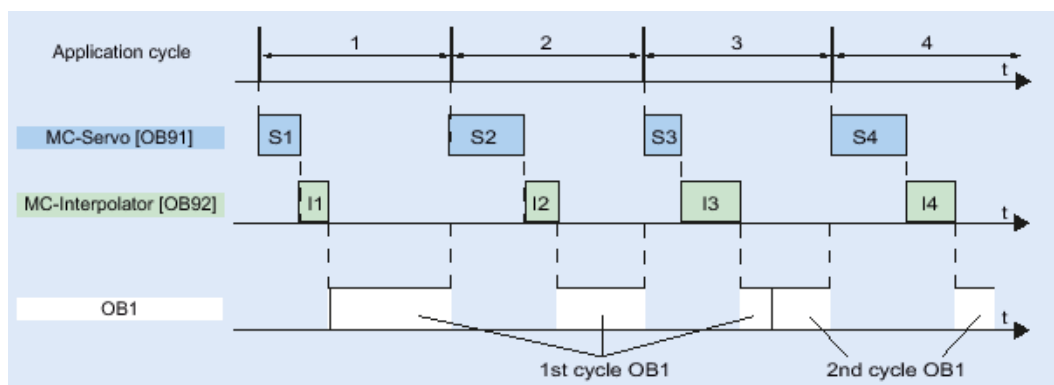


Figure 3.5: OB cycle priorities.

If the set application cycle is not adhered to, for example, because the application cycle is too short, overflows can occur. An overflow of the MC-Servo [OB91], MC-Interpolator [OB92], MC-PreServo [OB67] and MC-PostServo [OB95] is entered in the diagnostic buffer of the CPU and results in setting the CPU to STOP. MC-PreServo, MC-Servo, MC-PostServo and MC-Interpolator are stopped. If necessary, you can evaluate the entry in the diagnostic buffer via a time error OB (OB80).

---

## 3.3 Drive unit

### 3.3.1 Hardware definition

The hardware configuration of the machine is built inside the topology, the network and device view of the project tree. There, the physical modules of the machine are selected, defined and configured and the connection is established between them.

The topology view is a working area of the hardware and network editor. The main task here is to establish the way the PLC sends and receive information. The construction, for example, of a ring that exits from one port of the PLC end enters in the other one must be managed otherwise it does not know where the data must be sent. The following tasks in topology view can be made:

- Displaying the Ethernet topology
- Configuring the Ethernet topology
- Identify and minimize differences between the desired and actual topology

The graphic area of the topology view displays Ethernet modules with their appropriate ports and port connections. Here, additional hardware objects can be added with Ethernet interfaces. The table area, instead, displays the Ethernet or PROFINET modules with their appropriate ports and port connections in a table. It corresponds to the network overview table in the network view, 3.7 on the following page.

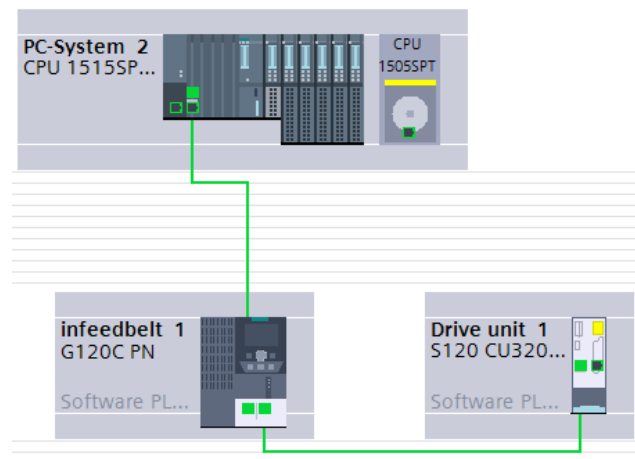


Figure 3.6: Topology view.

The network view shows the devices and their components with interfaces to the sub-nets. In connection mode, the communication paths of configured connections can be made visible. The following information about connections is then available:

- The devices between which connections have been created
- Which connection types have been created for a device
- The devices that have unspecified connections

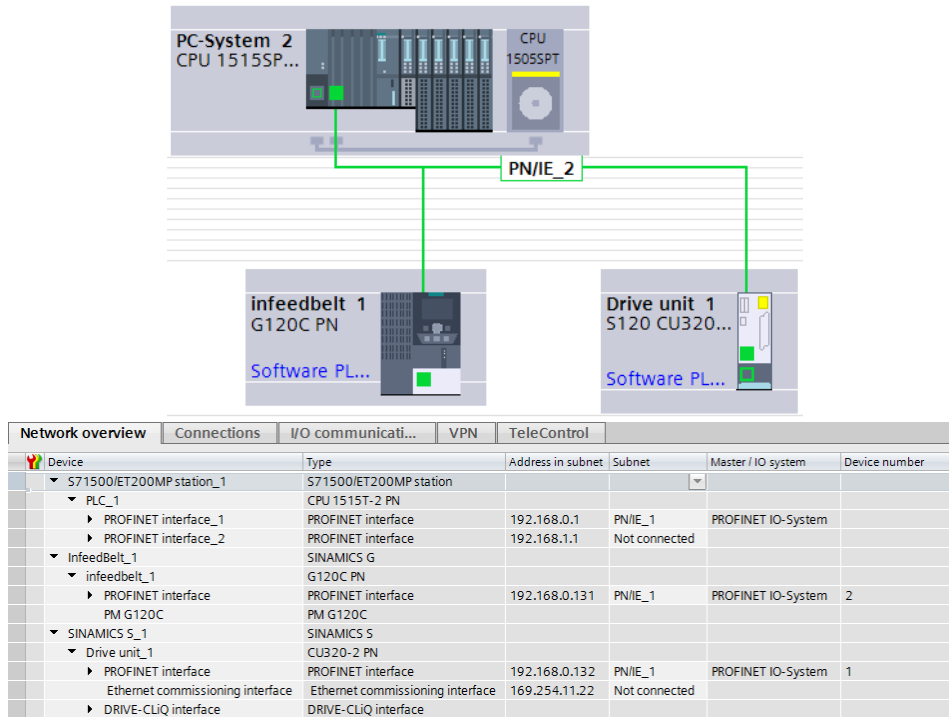


Figure 3.7: Network view.

The table area of the network view consists of several tables:

- **Network overview:** The table for the network overview supplements the graphical network view with the following functions: - detailed information on the structure and parameter settings of the devices. - Using the "Subnet" column, components capable of communication can be connected with created sub-nets.
- **Connections:** Communication connections are displayed in this table. Some aspects of the connections displayed here can be edited directly in the table. Selected connections are displayed in the Inspector window and can be edited there.
- **Relations:** Logical connections of the HMI client-server relation are displayed here.
- **I/O communication:** Device communication via PROFIBUS DP and PROFINET IO is configured and displayed in this table. In the I/O communication table, only those interfaces of a device are displayed that can be used for communication via PROFIBUS DP or PROFINET IO.
- **VPN:** Properties of the VPN user groups and their assigned security modules are displayed here.



---

The device view is one of three working areas of the hardware and network editor. The following tasks can be made:

- Configuring and assign device parameters
- Configuring and assign module parameters
- Editing the names of devices and modules

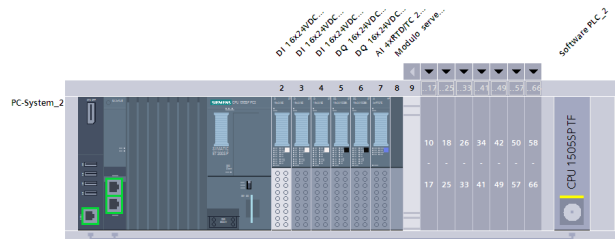


Figure 3.8: Device view of the PLC.

The graphic area of the device view displays hardware components and if necessary the associated modules that are assigned to each other via one or more racks. In the case of devices with racks (for example the PLC structure), there is the option of installing additional hardware objects from the hardware catalogue into the slots on the racks. The table area of the device view gives you an overview of the modules used and the most important technical and organizational data.

The hardware components presented and discussed in the previous chapter are added. Starting from the PLC, Fig. 3.8 shows the device view of its configuration. As said, the one chose is the ET200sp Open Controller where the input-output modules of the old PLC are inserted. The motion drives are attached to it. On one side, there are the infeed belt actuators already present in the previous machine. On the other, the new Siemens motors are inserted. The network is created in the network view 3.7 on the preceding page, where the hardware posts are configured in order to share the same IP address. The port to be used, instead, is specified in the topology view, 3.6 on page 78.

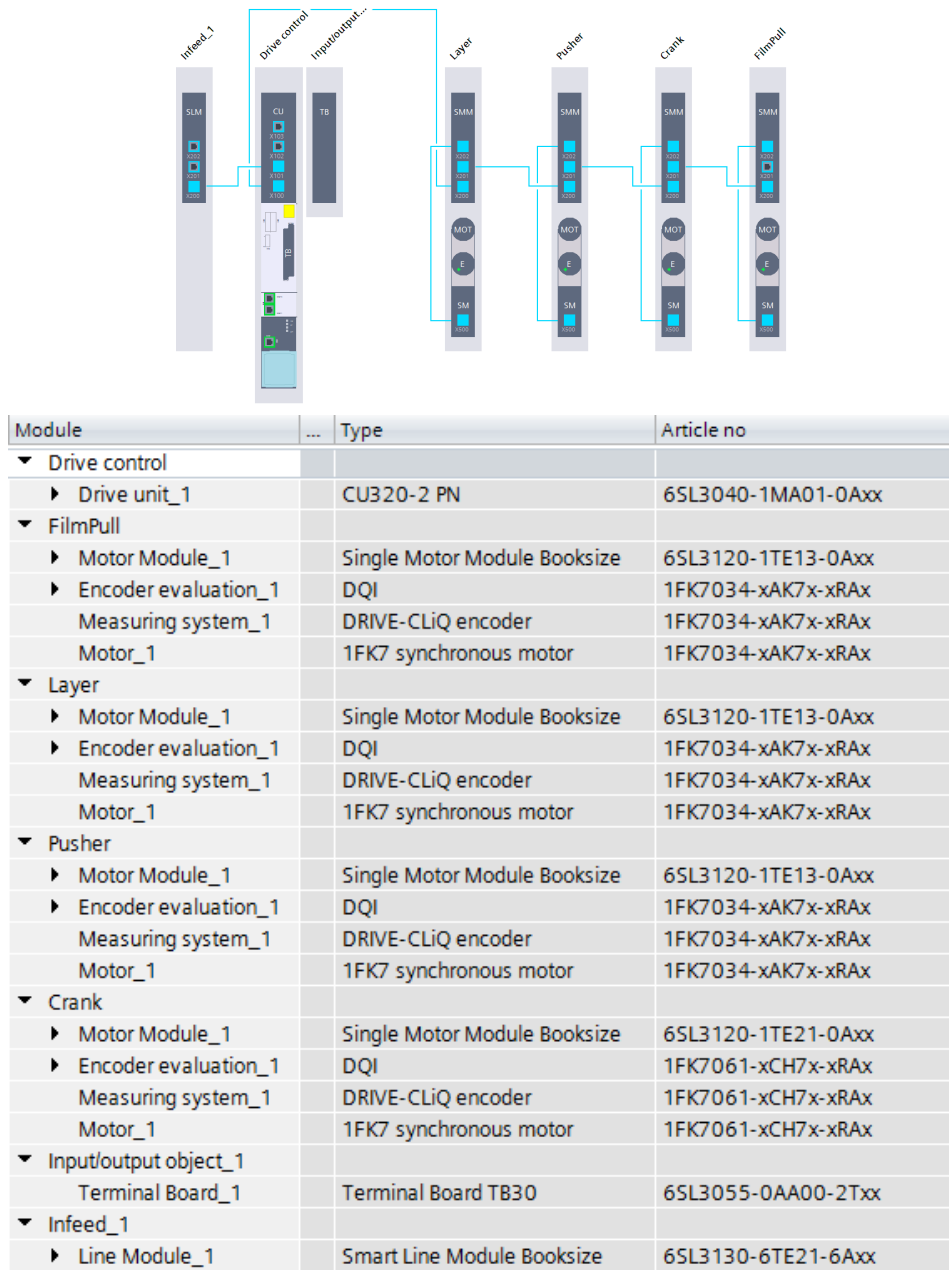


Figure 3.9: Device view of the drive.

The configuration is created in the "Device view". This is where the individual components are inserted and connected. Fig. 3.9 shows both the graphical implementation and the list of the used components. In particular, regarding the discussion on the new motor choices developed in the second chapter, the equivalent to the Schneider motors are inserted. For the configuration, firstly the drive S120 is inserted and connected to the PLC in the network view by configuring the IP address. Then, passing to the device view of the drive, four distinct motor modules are added. Their configuration consists of the selection of the type of module; namely, the characteristics that it should be able to give to the related motor connected. For the project purposes, their parameters are reported in the table below.

	Rated power	Rated current	supply voltage
Layer	1.60 kW	3.00 Arms	510-620 VDC
Pusher	1.60 kW	3.00 Arms	510-620 VDC
Film pull	1.60 kW	3.00 Arms	510-620 VDC
Crank	4.80 kW	9.00 Arms	510-620 VDC

Then, the desired motor is selected from the hardware catalogue and assigned to the relative module. This comes with an automatic connection of the inputs between them. In addition to this, the telegrams that establish a communication politics with the software must be specified. The standard choice is the telegram 105 how Fig. 3.10 shows.

Name	Item	Link	Telegram	Length	Extension	...	Type	Partner	Partner data area
▼ Drive control-Telegrams	1								
Send (Actual value)		➤	Free telegram	1 words	---	→	CD	Software PLC_2	E 256...257
Receive (Setpoint)		➤	Free telegram	1 words	---	←	CD	Software PLC_2	A 256...257
<Add telegram>									
▼ Layer-Telegrams	2								
Send (Actual value)		➤	SIEMENS telegram 105	10 words	0 words	→	CD	Software PLC_2	E 258...277
Receive (Setpoint)		➤	SIEMENS telegram 105	10 words	0 words	←	CD	Software PLC_2	A 258...277
<Add telegram>									
▼ Pusher-Telegrams	3								
Send (Actual value)		➤	SIEMENS telegram 105	10 words	0 words	→	CD	Software PLC_2	E 278...297
Receive (Setpoint)		➤	SIEMENS telegram 105	10 words	0 words	←	CD	Software PLC_2	A 278...297
<Add telegram>									
▼ Crank-Telegrams	4								
Send (Actual value)		➤	SIEMENS telegram 105	10 words	0 words	→	CD	Software PLC_2	E 298...317
Receive (Setpoint)		➤	SIEMENS telegram 105	10 words	0 words	←	CD	Software PLC_2	A 298...317
<Add telegram>									
▼ FilmPull-Telegrams	5								
Send (Actual value)		➤	SIEMENS telegram 105	10 words	0 words	→	CD	Software PLC_2	E 318...337
Receive (Setpoint)		➤	SIEMENS telegram 105	10 words	0 words	←	CD	Software PLC_2	A 318...337
<Add telegram>									
▼ Input/output object_1-Telegr...	6								
Send (Actual value)		➤	Free telegram	1 words	---	→	CD	Software PLC_2	E 338...339
Receive (Setpoint)		➤	Free telegram	1 words	---	←	CD	Software PLC_2	A 338...339
<Add telegram>									
▼ Infeed_1-Telegrams	7								
Send (Actual value)		➤	Free telegram	1 words	---	→	CD	Software PLC_2	E 340...341
Receive (Setpoint)		➤	Free telegram	1 words	---	←	CD	Software PLC_2	A 340...341

Figure 3.10: Modules Telegram.

Fig. 3.9 on the preceding page shows the list of components implemented at the end. An additional step consists of the activation of the IRT (Isochronous Real Time) mode. The fast and reliable reaction time of a system operating in isochronous mode is based on the fact that all data are provided just-in-time. The basis for this is a constant bus cycle. The isochronous mode function guarantees synchronization of the following at constant time intervals:

- Signal acquisition and output by the central and distributed I/O
- Signal transmission via backplane bus, PROFINET IO or PROFIBUS DP
- Program execution in the CPU in local time, in time with the constant bus cycle time

The result is a system that acquires its input signals, processes them and outputs the output signals at constant time intervals. Isochronous mode guarantees precisely reproducible and defined process reaction times as well as constant bus cycle and synchronous signal processing for central and distributed I/O.

---

To exploit this functionality, the modality must be enabled in the S120 drive. Moreover, in the MC-Servo OB automatically created by the software when a TO axis is added, its cycle must be set. The "synchronous to the bus" mode must be selected allowing the OB execution to be synchronous with the network. Then, the interval of the call must be also specified.

At this point, the motors generated must be assigned to the TO axis of the project. This is performed by specifying in the TO configuration the encoder that the axis is going to move.

### 3.3.2 Parameters access

Once the configuration of the hardware is terminated, it is necessary to understand how to access their information. This is required, in particular, for the managing and the displaying of the faults.

In Fig. 3.10 on the previous page it is possible to appreciate the ordering number assigned to each component. They are assigned depending on the order of creation of the components. From them, additional variables related to the newly added components are generated. They represent the addresses that point to the physical components allowing access to their internal variables. This is what is going to be used for the managing of the faults activation explained in the fourth chapter. In addition to this, another aspect must be considered. The modules are characterized by parameters that act as inputs and outputs of the module. In this way, it is possible to use those of interest. The two used are:

- p2111 tells the number of alarms active since the last reset;
- p3981 is an acknowledge bit that allows to reset the alarms active in the module when it is set to one. Then, it returns to zero at the end of the acknowledge.

By taking advantage of them, you can perform error handling. They allow us to see only the active number of faults and to reset them. In order to see the information about the active ones additional parameters are used:

- r2131 displays the code of the oldest active fault;
- r2132 displays the code of the last alarm that occurred.

They are accessed directly by the HMI panel which reads the information available. To do this the parameter address of interest must be built. It comprises parameter number, index and DB number:

*DB<parameter number>.DB<a>data block offset*

- The data block number corresponds to the parameter number.

- the data block offset is formed from the DO number and the parameter index as follows:

*Data block offset (binary):*  $\underline{x_{15} x x x x x_{10} x_9 x x x x x x x x_0}$

The first six digits refer to the drive object number, while the last ten is the parameter index. In detail, the offset is built in this way:

*Data block offset = 1024\*drive object No. + parameter index*

- the <a> value is assigned as a B, D or W depending on the parameter type

Data type parameter	Data type HMI variable	Offset
Integer8	Byte	B
Integer16	Int / Word	W
Unsigned8	Byte	B
Unsigned16	Int / Word	W
Unsigned32	DInt / DWord	D
FloatingPoint32	Real	D

As a result the addresses used in the HMI panel dedicated to the the drive error are:

Axis	Parameter 2131	Parameter 2132
Layer	DB2131.DBW2048	DB2132.DBW2048
Pusher	DB2131.DBW3072	DB2132.DBW3072
Film pull	DB2131.DBW4096	DB2132.DBW4096
Crank	DB2131.DBW5120	DB2132.DBW5120

---

## 3.4 Libraries

The *Libraries* folder contains 4 different pre-made Siemens Libraries. They can be added through the Global Library section in the TIA Portal software. Each library contains in addition to the blocks also the tags and the data types that must be included in the relative project folders.

### 3.4.1 IsAxisReady

This library allows to safely and correctly activate the drives of the machine. This is a really useful tool because before a technology object can be enabled without error via "MC\_Power", all the necessary sensors (encoders) and actuators (drives) must be available for the technology object. Below is explained the procedure that must be done in order to successfully enable the TO. Afterwards, the library itself is reported.

*Description:*

Using the Motion Control instruction "MC\_POWER" it can be enabled or locked a technology object via the "Input" parameter. Before setting the "Enable" parameter to True, the parameterized sensors (encoders) and actuators (drives) should be available for the technology object.

The following requirements must be met to enable the technology object with the MC\_Power instruction:

- The technology object has been correctly configured.
- The cyclic BUS communication has been established between controller and sensor (TO.StatusSensor[.].State).

Since the CPU usually starts up faster than the connected IOs, in most cases the technology object cannot be enabled immediately after the CPU startup. Data transfer (online) can only happen if the communication connection between CPU and drive has been established.

The complete availability of a technology object can be checked via the variable "CommunicationOK" in the following structured data types:

- Sensors (encoders): "TO\_Struct\_StatusSensor"
- Actuators (drives): "TO\_Struct\_StatusDrive"

The variable "CommunicationOK" does not confirm the availability of a technology object, but whether a cyclic communication has been established between the technology object and the drive. This means that the availability of the connected drives (actuators) and encoders (sensors) must be controlled.

*Special features:*

- TO\_ExternalEncoder: There is no actuator (drive) on this technology object.
- TO\_SpeedAxis: There is no sensor (encoder) on this technology object.

The following figure shows the details view when the "DB Editor" technology object of a "TO\_PositioningAxis" is selected in the Project Tree.

▪ ▼ StatusSensor	Array[1..4] of TO_Struct_StatusSensor
▪ ▼ StatusSensor[1]	TO_Struct_StatusSensor
▪ State	DInt
▪ CommunicationOK	Bool
▪ Error	Bool
▪ AbsEncoderOffset	LReal
▪ Control	Bool
▪ Position	LReal
▪ Velocity	LReal
▪ AdaptionState	DInt
▪ ▶ StatusSensor[2]	TO_Struct_StatusSensor
▪ ▶ StatusSensor[3]	TO_Struct_StatusSensor
▪ ▶ StatusSensor[4]	TO_Struct_StatusSensor

Figure 3.11: TO\_Axes Sensors.

The status of the variable "CommunicationOK" shows you whether the encoder or drive is ready for cyclic communication with the technology object. If the status value is not True, the cause is displayed in the diagnostics buffer.

*Example for safely triggering "MC\_Power":*

Triggering of the input "Enable" of the Motion Control instruction "MC\_Power" can then be done as Fig. below shoes. In particular, the code in Fig. 3.12 on the following page reports that the technology object is enabled only when the user enable signal is set and the corresponding sensor (encoder) and actuator (drive) are available for the technology object.

*Note:*

If the technology object is enabled via the "MC\_Power" instruction before the sensor and actuator interfaces ("CommunicationOK") receive the True signal, then, depending on the configuration, there might be alarm messages like "Encoder driver/actuator driver not initialized during startup" or "Adjustment cannot be restored", for example.

The "IsAxisReady" function block is used to check whether a technology object can be enabled or referenced. This function block checks the named criteria and returns the value 'True' if the technology object is ready for use. This can be used for the start of "MC\_Power".

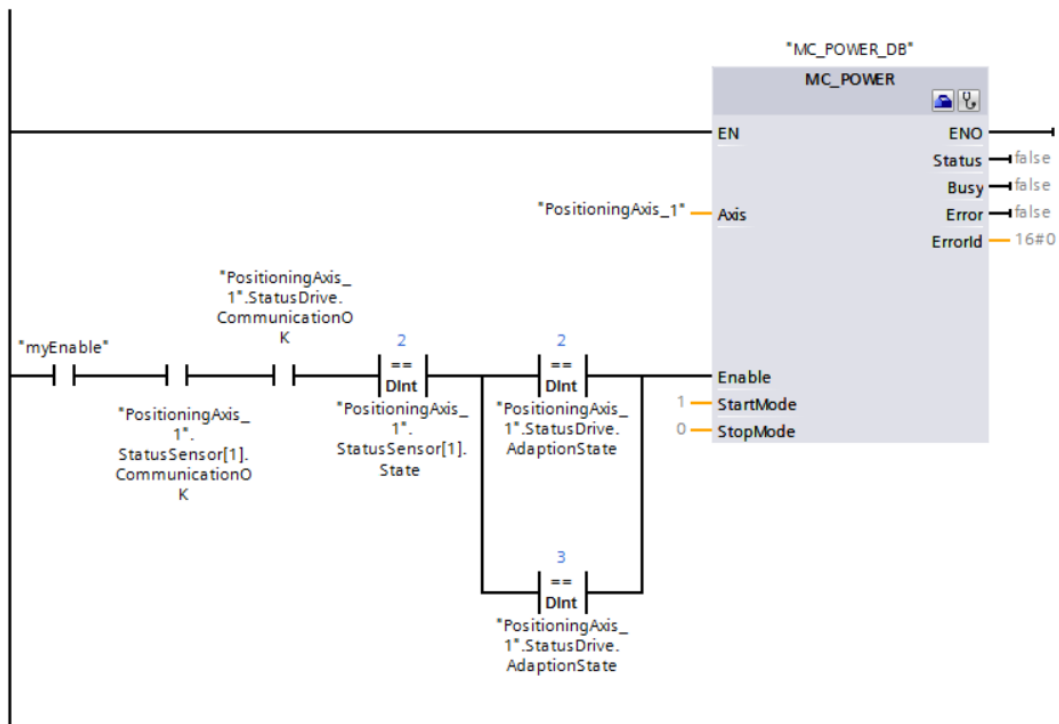


Figure 3.12: Example of Enable activation for MS\_Power.

For checking, the Motion Control instructions "MC.Power" and "MC.Home" can be linked with the "Enable/Execute" signal. These instructions are executed only if the technology object is ready for use. As an additional criterion, the input parameter "includeActorSensorErrorCheck" can be set to True on the "IsAxisReady" function block. In this way, the actuator and sensor are checked in addition for faults. Note that with the interconnection with "MC.Power.Enable" the axis enable might be removed in the case of a fault. In the case of motors without holding brake there is then a risk of diminution of the load. The library used in the project includes the "IsAxisReady" function block for the technology objects "TO.SpeedAxis", "TO.ExternalEncoder", "TO.PositioningAxis" and "TO.SynchronousAxis". In this case, it is exploited for "TO.PositioningAxis" and "TO.SynchronousAxis" only.

### 3.4.2 LAxisCtrl

The library LAxisCtrl is the centre of the Siemens motion control software structure. It implements FBs for simple and central control of the basic motion control functions of axes (technology objects) of a SIMATIC S7-1500 or S7-1500T. The central view of each axis via this standard application enables easy programming, fast commissioning and direct testing of the project.

The standard application with the blocks of the "LAxisCtrl" library enables to easily control extensive motion control functions of the axis type technology objects. By viewing the axis via a central block, programming and commissioning of axes can be made considerably easier. Technological peculiarities,



---

e.g. when replacing cam disc synchronization, are taken into account internally in the module. A separate function module is available for each axis type (speed, positioning, synchronous axis, external encoder), whose interface is respectively tailored to the available functions of the axis type. The axis blocks, therefore, contain the sum of the inputs and outputs of the integrated PLCopen blocks. The uniform configuration of the different axis blocks makes it easy to switch between the different axis types.

Due to the extensive functionality, the configuration of the commands, e.g. for dynamics and position specifications, is not via individual block inputs, but via a variable structure of the "LAxisCtrl\_typeAxisConfig" data type.

The "AxisCtrl" axis function block is also used as a basis in higher-level technological software modules, such as multi belt control, which are published as industry-specific standard applications in Industry Online Support.

**Explanation of the blocks** The interfaces and the controlling of the LAxisCtrl function blocks are based on the PLCopen standard with taking into account the behavior of executing and enable inputs. The function blocks are implemented in Structured Control Language (SCL). They are programmed for use in a cyclic task. These blocks implemented acts as a "General Axis" FB for the TO\_Axis dedicated. Indeed, in parallel to the logic of managing for the FB, all the motion Control FBs available by Siemens are present for the compatible axis FB contained in the library. This allows a faster managing and control of the axes used in the code.

*Activation and deactivation of the function blocks:*

The axis function blocks are activated via the enable input. This input is level sensitive. With a falling edge the functionality of the FB is stopped and the technology object (TO) will be disabled if it is active.

*Selection of the functionality:*

The functionalities of the axis function blocks are divided into three groups:

- General axis functions (enable axis, reset axis, passive/direct homing, torque limiting)
- Functions in the basic coordinate system (positioning, move axis at velocity set-point, jogging, stop, fast stop, active homing, all synchronous motion commands)
- Functions in the superimposed coordinate system (superimposed positioning)

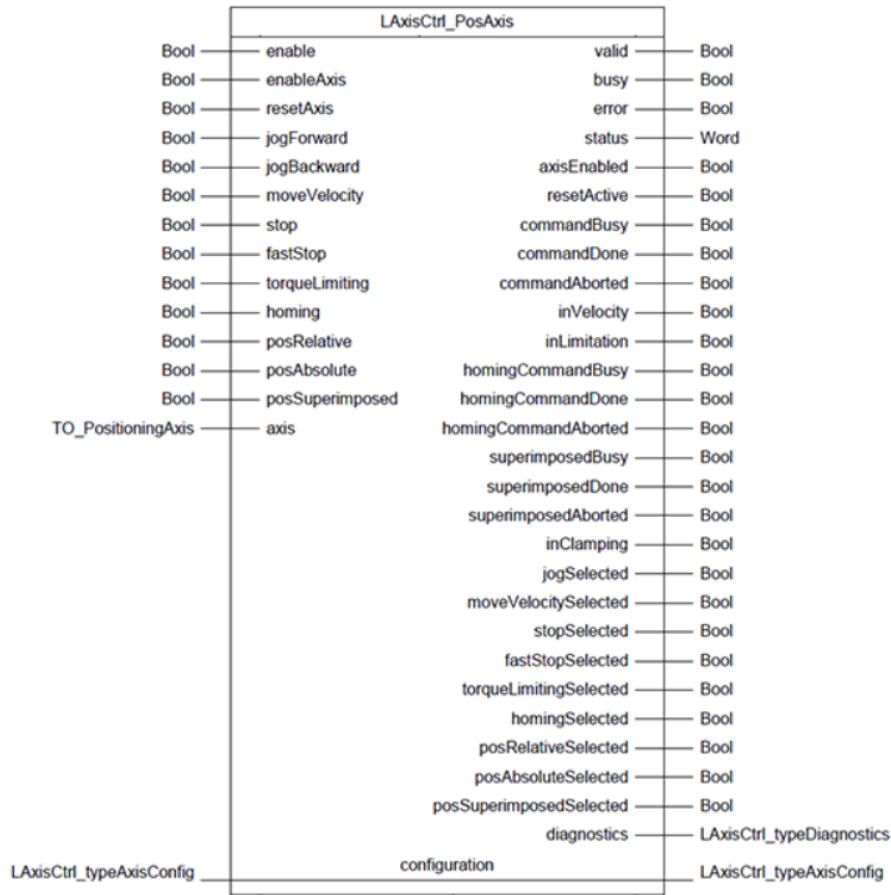


Figure 3.13: Example of LAxisCtrl FB.

To use the commands a configuration is necessary, e.g. for dynamics or positions. The configuration is done by adding a variable with `LAxisCtrl_typeAxisConfig` data type in a memory section and connecting it to an instance of an axis control function block. In this way, by accessing to these structured data, it is possible to modify the behaviour of the axis. If the inputs of the FBs allow to start the operation requested, through this variable is possible to affect that very operation. For example, by modifying the set-points of position, velocity and acceleration it is possible to obtain different `Move_Velocity` or `Pos_Absolute`. Another example is the modification of how the `TO_SynchronousAxis` executed the cam coupling with the master, simply by changing how the cam is activated. This structure contains all the parameters that are set as input for every Motion Control FB implemented in the TIA Portal software.

In addition to the mere control of the axes, this library adds also some status control FBs. Indeed, if the axes FBs already have a diagnostic output for both errors and state of the axis, some additional and specific FBs are present. For example, it is the case of the `LAxisCtrl_GetAxisStatusWord` which gives direct access to the decomposed bits of the axis "Status Word". This allows a simpler on-line analysis during the motion control. Furthermore, a different but very useful FB is represented by the `LAxisCtrl_BrakeControl`.

---

The previous FBs are used to act on the technology object defined in the project. So, the self-generated OB MC-Servo communicates with the physical drives. Instead, to allow access on the drives and the actuator this latter FB allows direct control of the motor brake to enable or disable it.

### 3.4.3 LCamHdl

Motion sequences in modern production machines are realized with electronic cam disks instead of former mechanical cam disks. Advantages are optimization possibilities regarding jerk-free movements, maximum acceleration, reduced vibrations, reduced wear and reduced downtime when changing formats. The creation, as well as the modification of the cam disk, should also be possible during the normal operation of the machine to adapt to the current product. This shall be as simple and performant as possible.

The LCamHdl library for SIMATIC and SIMOTION provides function blocks that support the user in creating high-quality and jerk-free cam disks. The necessary calculations of the segments of different profile types especially of polynomial coefficients and standardizations are done by the function blocks. This library introduces different ways to implement a cam profile in the software.

**LCamHdl\_CreateCamBasic** A fully defined cam disk shall be created at runtime. Points in the cam disk and the according dynamics are known. Transitions can be made via straight lines and 5<sup>th</sup> degree polynomials, taken into account velocity and acceleration. In this case, the CreateCamBasic function block eases the cam disk creation for cam disks with interpolation algorithms up to 5<sup>th</sup> degree polynomials. The leading value of the first point defines the start of the cam disk and the leading value of the last point defines the end of the cam disk. As the indexes in the cam profile increase also the according leading values have to increase. A maximum number of 51 points can be used in a cam profile to define a cam. The FB CreateCamBasic first fills the necessary segments in the cam technology object and then interpolates the cam.

**CamHdl\_CreateCamAdvanced** The function block CreateCamAdvanced can be used to merge working ranges and motion transitions into one cam disk at run-time. Unlike directly assigning the cam's data block, the FB can be used without having to calculate the polynomial coefficients before. The FB is based on the motion rules for cam mechanisms. The cam profile configuration of the position as well as the geometric derivations is made in the real section (e.g. velocity, acceleration, jerk). There are different mathematical functions available for the motion transitions (elements), subsequently called profile types. Besides polynomials of 3<sup>rd</sup>, 5<sup>th</sup> and 7<sup>th</sup> degree further profiles can be implemented such as: straight line, quadratic parabola, basic sine, modified acceleration trapezoid, modified sine, etc. In addition to that, it is also possible to transfer single points, which makes it possible to generate cam disks

---

with combined ranges consisting of transition functions and of single points.

Differently from the `CreateCamBasic` block the function block `CreateCamAdvanced` works segment based. This allows gaps (filled by run-time interpolation) between segments and also the usage of the points array in the cam technology object. So, the FB `CreateCamAdvanced` fills the necessary segments and points in the cam technology object starting at `startSegmentIndex` and `startPointIndex`. A maximum number of 50 segments and 1000 points in a cam profile can be used to define a cam. The function block can be configured to delete preceding or successive cam points and cam segments. In addition it is possible to interpolate the cam disk at the end with the "interpolateCam" configuration bit.

**LCamHdl\_CreateCamBasedOnXYPoints** A cam disk based on interpolation points is to be created at run-time. Only the X and Y coordinates of the interpolation points are known (where X defines the master position and Y the slave one). In this case should be used the `CreateCamBasedOnXYPoints` function block. It eases the cam disk creation for cam disks consisting of just interpolation points. The interpolation mode (linear/C splines/B splines) can be defined through the FB settings. By exploiting this FB a maximum number of 1000 points can be used in a cam profile to define a cam.

Several cams are used in the Motion Control project as part of the represented technology object. To manage them easily and at run-time you need to use this library. Indeed, one fundamental part of the motion control is the Format change. As it is presented later, this requires an online update of the cam points and a successive interpolation. This procedure is not possible without using the library shown. In particular, the most exploited FB is the `LCamHdl_CreateCamAdvanced`. It is used for almost all the cam creation of the project. In the case of rotary and linear cams acting on the complex crank structure mentioned earlier in this chapter, the point-to-point conversion of the rotary cam requires the use of the `LCamHdl_CreateCamBasedOnXYPoints`. Indeed, as the following library section will explain, the newly created master-slave positions must be merged and this FB solve the problem.

### 3.4.4 Crank module

This library is a custom set of FB and FCs created to perform the conversion between the linear and the rotating motions of a crank mechanism. This comes from the necessity to convert the linear cam provided for the motion of the sealer axis with the cam of the motor that should move it. Indeed, this operation in the Schneider algorithm is performed through native software functionalities, allowing an automatic conversion between the two worlds. In Siemens, such a possibility is not available, making it difficult to use the provided linear cams in the motion control.

---

This library is developed after the consideration of different possible solutions:

- online conversion inside the MC-PreServo of the position setpoint that the linear cam gives in relation to the master degree.
- offline creation of a table of positions then used online for the matching of the cam positions
- development of an offline polynomial conversion for the creation of the rotating cam from the linear positions

The first solution is more intuitive. The software controls the motors by sending a position target created by the motion control logic to the drive which automatically converts it to speed control. Therefore, the online conversion of the linear position to the rotating degree could be a good solution. Unfortunately, the servo cycle must be as small as possible, making long computations not particularly suitable. This solution, then, could become a source of overflow problems for the cycle.

The second solution overcomes the problem that can arise in the previous scenario. A table computed offline makes the matching of the linear position into a rotating one much faster. However, this would imply the exact knowledge of the value of each linear position that is in the linear cam profile. The table or more specifically the array should be very long and not reliable in case of large changes in the cam profile.

The last solution merges the positive aspect of both the previous without introducing particular problems. The cam computed offline allows the fast and reactive tracking of the rotating profile. Therefore, this library consists of essential functions which are called inside each other. The structure is composed of the main FB called *Poly5\_360\_crank* which allows converting the cam profile. This one exploits the computations performed by the FC *Crank\_Translations* which contains the two FCs that actually compute the conversion: *TranslationToRotation\_Compact* which computes the angles from the linear position and *RotationToTranslation\_Compact* which computes the linear displacement from the angle. Starting from the last ones, they are built in a similar way. They are divided in parts that allow computing different parameters from the inputs. Apart from the value to be converted, both have as input the geometry parameters of the crank which are passed through a custom structure variable composed of:

- Radius length
- Rod length
- Start offset
- Vertical offset
- Window

These parameters are shown in Fig. 3.14 on the next page where it is possible to appreciate their meaning. The two lengths are the main structure of the crank mechanism. The two offsets represent respectively the initial

position and the displacement between the center of the motor and the level of motion for the rod. These are the mechanical parameters of the machine. The Window instead is an arbitrary section of angles around the dead centres; in the case of the MS260, due to geometry, they are around  $0^\circ$  and  $180^\circ$ .

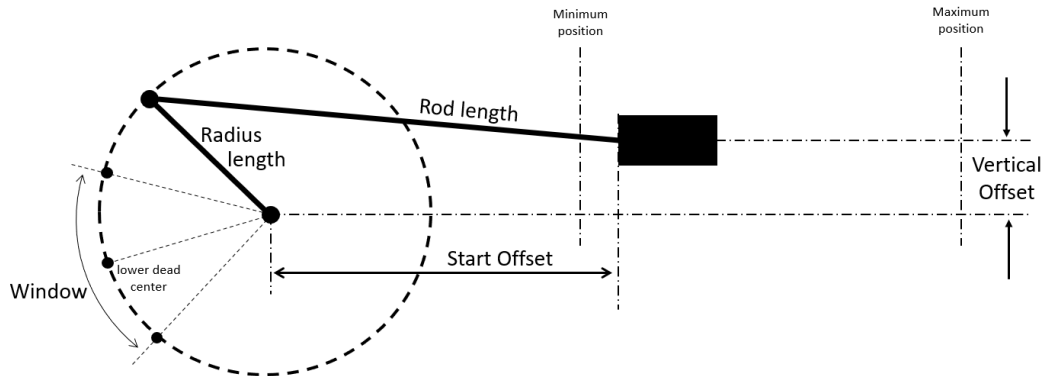


Figure 3.14: Crank module structure.

**Motion conversion FCs** The *TranslationToRotation\_Compact* allows obtaining the two angles that coincide with the linear position. In addition, the two extremes of the linear motion are computed. These are performed in different parts of the code. Firstly the linear position is computed adding the starting position. Then, the check of the mechanical components is performed. In case one of them is a negative value or the vertical offset is lower than the sum of the rod and the radius, an error is raised. Then, the first part is dedicated to the computation of the extremes of the linear motion. To do this, the relation of the length of rod and radius is performed in order to understand the mechanical structure. For this reason, different formulas can be used. Basically, they rely on Pythagoras's theorem. Next, the two angles are computed. The previous parameters are used to identify the truthfulness of the linear position. In case it exceeds one of the two extremes, an error is raised. If this does not happen, the temporary angles of the system are computed. In particular, the hypotenuse between the linear point position and the centre of the crank computed through the Pythagoras's theorem is used for Carnot's theorem. Consequently, thanks to them and to the mechanical parameters, it is possible to identify different scenarios of the crank configuration, making it possible to find the correct angles. In Appendix A, the code implemented is reported.

The parallel block, the *RotationToTranslation\_Compact*, is organised in a similar way. After the initial code for the check of the input parameters, the extreme angles are defined. Also in this case, using the geometry of the system it is possible to find the two angles. In this case, instead, the theorem of sine and cosine applied to the triangle are exploited. The two angles are then used for the principal goal of finding the linear displacement. Firstly, they define a range in which the input angle must be contained, otherwise an error is raised. Then, the formula of the crank system is applied in order to find the linear

---

position. The final value is controlled in order to be sure that it is greater than zero. As before, Appendix A contains the code implemented.

Coming to the next FC, the *Crank\_Translations* allows to call the previous two by merging their computations by having common inputs and outputs that are passed from outside.

**Cam conversion computation FB** The last and more important function is the *Poly5\_360\_crank*. This FB is used in the motion control algorithm for the definition of the crank rotating cam profile. The inputs are the geometry of the mechanism, window of the dead centres and the segments of the cam profile. The latter is an array of a custom structure that allows defining all the parameters for the definition of the profile. Each segment is characterized by the start and end position of the master, along with its velocity, and the start and end values of the position, velocity and acceleration of the slave. All of these information are necessary for the definition of the cam segments. Indeed, this FB is based on the application of a polynomial function of 5<sup>th</sup> order. Giving the extremes of a segment, it is possible to find the resulting function that merges the two points. The FB must be called inside a "for cycle" in order to compute the profile of each segment. The *Cam\_Track* output of the block is a particular custom structure. It contains the structure "Points" and two integers (*LastIndex\_Linear* and *LastIndex\_Rotative*) which define the last elements used in the linear and rotating arrays. The structure "Points", instead, contains two different arrays (one for the linear and one for the rotating cam) of type *LCamHdl\_typeXYPoint* which generates an "X" and "Y" element for each point of the array. This definition implies the use of the *LCamHdl\_CreateCamBasedOnXYPoints* FB for the interpolation of the cams. As the LCamHdl dedicated section describes, this FB works on the interpolation of single points. This is different with respect to the *LCamHdl\_CreateCamAdvanced* used for the other cams. In this case, the profiles come from the definition of points on the base of a polynomial function.

The code starts with the definition of the slopes of velocity and acceleration that are wanted for the segment extremes. They are computed, indeed, from the start and the end velocities and acceleration of the slave and the velocity of the master. Then, the coefficients of the polynomial are defined. They come from the computation of the interval (T) of the segment in terms of master degrees and the initial values computed. The former is computed from the master start and end positions defined for the segment. They are the key to the creation of the profile as below it is described. What follows is the effective computation of the points of the two output arrays.

The arrays are filled through the use of a "for cycle". It covers the interval (from zero to T) previously computed by increasing the index by one per cycle. This allows computing with an accuracy of one degree, relatively to the master positions, the profile of the cam. Then, each element of the output arrays is a point of the cam. The elements of the arrays to be filled follow the sum of the index and a shift value. This last one is always updated before the start of the cycle. In the case of the linear cam, this coincides with the master

---

start position. So, each cycle starts at the master start position (defined in the input segment) and ends at the master end position (sum of the "for" index and of the shift value). This results in the filling of 361 elements of the linear array (as the master degrees go from 0° to 360°). In the case of the rotating cam, the shift value is the last array index used in the "for cycle" of the previous segment. Obviously, for the first segment, it is modified to zero. This is possible because the master positions are always increasing in the cam profile. Therefore, they allow to safely define the elements of the arrays to be filled.

Once it is clear how the arrays are filled, the computation of the proper points follows. Thanks to the "for" index it is possible to compute the value of the polynomial function. Indeed, the parameters are defined at the beginning of the FB with the whole interval length. So, the single index defines the value of the function inside that interval T. This value computed is then added to the Y element of the linear array, while the X is filled with the correspondent master degree. At this point, the rotating array is filled. Firstly, the Y point is converted to the correspondent angle by the *Crank\_Translations* FC. This allows filling the rotating array with the angle found in the Y element of the rotating array, while in the X is set the master degree. Lastly, the window parameter is finally used. The dead centres of the crank are critical points, where large variations of the degrees are matched with little variations of the linear displacement. Therefore, the conversion from linear position to angle became useless. For this reason, the window defines a range of angles where the crank conversion is not applied. In case the angle is inside this window, it is actually not added to the array and a counter of points not stored is incremented. This is done in order to let the *LCamHdl\_CreateCamBasedOnXYPoints* FB to merge the points with the spline in order to let the rotating axis to be free with respect to the linear profile. The only exceptions are the first two and the last two elements of a segment. This allows to give the starting and ending directions of the spline in case the segment start or ends in the window. The most relevant case is when a constant cam profile is defined inside this range. Not adding all the elements would be a loss of information as no points would result. At the same time, only the first and the last points would not give enough information to the spline to maintain the constant profile. Therefore, two points at the beginning and at the end of a constant segment impose a sufficient amount of information to maintain the desired profile. The counter used to detect the points not stored is used in order to avoid empty spaces in the array. It allows decreasing the "for" index exclusively for the array element that must be filled along with the definition of the last index used at the end of the code.



# Chapter 4

## Motion Control algorithm

In this chapter, the motion control code developed is analysed. The *Motion* folder showed in Fig. 3.1 on page 69 contains the actual algorithm for the motion control. It is divided into folders and FBs for a better organization of the code. The blocks are related to sections of the algorithm in common with all the sub-modules. The two main parts are represented by the folders *SM\_Layering*, namely the sub-module dedicated to the first part of the machine, and the *SM\_Stretch* dedicated to the second part where the sealing is performed. The *Libraries* folder (already discussed in the previous chapter) and the *FC/FB\_Uutilities* folder (covered with the explanation of the code) are added to them. After an introductory section where the project choices and the main piece of code are explained, the following ones cover the actual code developed.

### 4.1 Implementation of the motion control

Starting from the main programs executed by the PLC, the OB blocks are the initial step to be considered. Organization blocks (OBs) are the interface between the CPU operating system and the user program. They are used to execute specific program sections: at CPU startup, in cyclic or clocked execution, whenever errors occur, whenever hardware interrupts occur. Organization blocks are executed according to the priority they have been allocated. Regarding those present in the project, they are of two categories. One of them contains OBs linked to the technology objects and they are treated in the section above dedicated. Instead, the other OBs are the main ones executed by the CPU cyclically, every time they are terminated, or by call. Among these, the one of interest is the *Block management* or OB1. The operating system of the S7 CPU executes OB 1 cyclically. When it has been executed, the operating system starts executing OB1 again. Namely, a cyclic execution is started after the startup has been completed. OB1 has the lowest priority of all OBs that are monitored for run-time. With the exception of OB90 all other OBs can interrupt its execution. The following events cause the operating system to call OB 1: end of startup processing, the execution of OB1 (the previous cycle) has finished. When OB1 has been executed, the operating system sends

---

global data. Before restarting it, the operating system writes the process image output to the output modules, updates the process image input, and receives any global data for the CPU. S7 monitors the maximum cycle time, ensuring a maximum response time. If the program exceeds the maximum cycle time for OB1, then the operating system calls OB80 (time error OB). If OB 80 has not been programmed, the CPU changes to STOP mode. Apart from monitoring the maximum cycle time, it is also possible to guarantee a minimum cycle time. The operating system will delay the start of a new cycle (writing of the process image output to the output modules) until the minimum cycle time has been reached.

The OBs were already present in the Siemens-Schneider version of the MS260 and they are kept. This comes from the project choice to maintaining the modular structure that was present in the old version. Indeed, in this case, the motion control is not incorporated in the PLC status control of the machine that was communicating with the Schneider drive. Instead, it is added as a modular section of the code that interfaces itself with communication bridges. Anyway, this approach is still different with respect to the previous one. As it is explained in the section dedicated to the communication between PLC and Drive of the first chapter, this requires more resources than a simple bridge. Indeed, it was explained that a word construction system is used in both codes for the internal variables used and in the PROFINET connection. Consequently, maintaining the modular structure, the new motion control can exchange information directly with the PLC variables stored in the *DB\_Axis\_IN* and *DB\_Axis\_OUT* previously used in the PROFINET network. By applying this to the PLC code, a reduction in the FCs called is achieved. Indeed, the parts eliminated from the main OB are related to the maintenance of the network and to the construction and deconstruction of the word passed. In its place, the new FBs computing the motion control are instead added.

## 4.2 Motion structure

The organization of the new code is composed of three main FBs called inside the OB 1 like the beginning of this chapter underlines. They represent the whole structure of the motion control and are divided into two layers as Fig. below shows.

Each module is coded using the Ladder language for a continuity line with the previous PLC organization. Indeed, this language allows a better debugging procedure with respect to a structured one. However, it is used only for the structure directly inside the three modules. Especially for the two sub-modules, many different FBs are present inside them. This code allows a visual disposition of the blocks and a better understanding of the active or faulty ones. Instead, inside each programmable block called by the two sub-modules the Structured Text language is used. The reason behind this choice is that the managing of the axes control is easier.

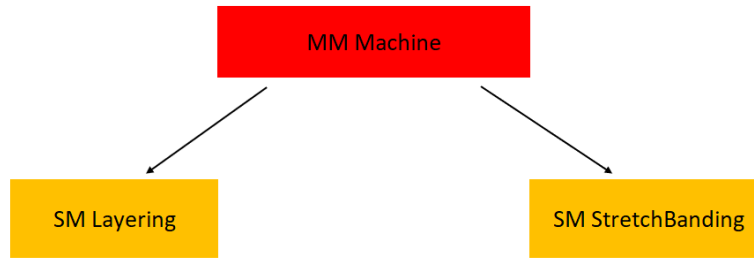


Figure 4.1: Siemens motion control layers.

### 4.2.1 MM\_Machine

Starting from the first of them, the MM\_Machine acts as an overlay level. It has the role of computing the instructions that are in common between the two sub-modules. Indeed, from this FB no commands are generated for the two sub-modules. They are directly obtained from the PLC variables through the bridge. Instead, the MM\_Machine performs a merging procedure of some outputs coming from the two sub-modules and allows to initialize some procedures. It is organised in six networks according to the Ladder structure specified above. In particular, four different jobs are here performed.

The first two networks combine the states of the two machine parts in order to notify a unique state to the status control part of the PLC. In particular, as will be shown later in the dedicated sections, the two sub-modules have in the outputs also the status of their operative conditions. So, instead of passing directly these outputs directly to the PLC variables, the MM\_Machine merges the states into a single variable in common between them.

An important common operation of the two sub-modules is the format change. This operation requires to compute the new cam of the axes from the data inserted in the HMI relative to the new package dimension and bundle data. To start, this operation needs the new dimensions of the bundle. To avoid the redundancy of their computation and of the data transferring, this is directly performed in this module. Then, the newly defined variables are passed to the sub-modules through the *stModulesInOut* common DB. Moreover, this first step of the format change allows recognizing the successful request from the HMI of a new format. Indeed, when this happens, the data inserted in the HMI are processed by the PLC. Only in case they are not faulty, they can be processed by the motion control. In this FB, the check control of the successful format change request is processed. In this way, the sub-modules which control the motion of the axes are not disturbed.

In parallel with the status management described above, also the errors that occurred in the two sub-modules of the motion control must be transmitted. For this reason, another job executed by this FB is related to alarms handling. The errors generated in the sub-modules are merged with the error

that eventually occurred in the drive. This is performed because the drive is a common component of the actuators and therefore of the sub-modules. Then, the control of its faults is performed inside these last networks of the MM\_Machine. The logic used for this operation relies on the use of the Sina-ParaS FB that is also used in the sub-modules for the actuators faults. The explanation of it is then described below. To avoid redundancies for the data transmission, the result of the reading procedure is directly added to the info displayed on the HMI. Indeed, after the drive error control, the sub-modules faults are taken from the "Modbus" common DB and the information are printed on the variables accessed by the HMI.

## 4.2.2 Sub-modules organization

The two sub-modules are structured in the same way between them. In particular, they are coded using the ladder language (with the advantages of this coding choice as previously mentioned) and organised in a SFC-like structure. This is slightly different with respect to the Schneider motion control described in the first chapter. Even if the parallel work performed by the different core functionalities is kept, the requirements and the steps to reach that point are different. Indeed, Fig. below summarizes the structural idea behind the choices made for the organization.

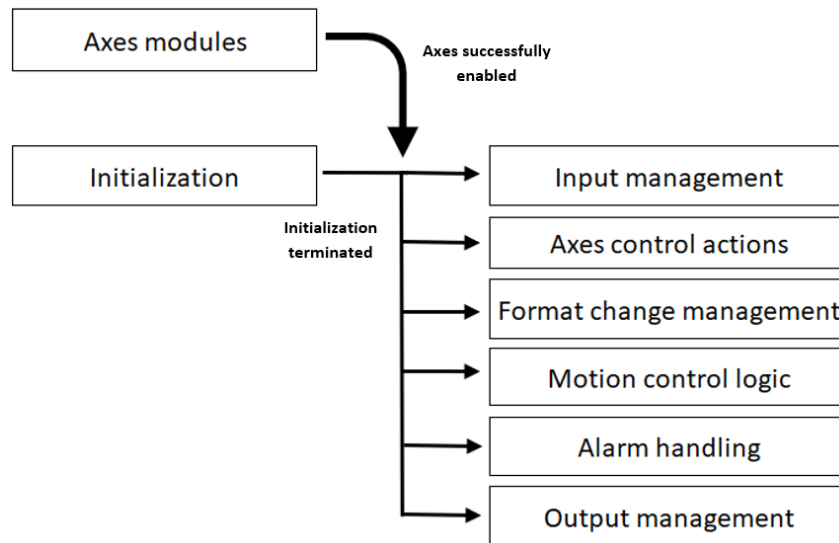


Figure 4.2: Sub-module Structure.

Each block reported here is a programmable block that can be either a FC or a FB. In particular, the Axes modules and the Axes control actions are a group of programmable blocks, but they are covered later. Regarding the two types of blocks mentioned they have significant differences and reasons to be used:

- A function block (FB) is a block "with memory." It is assigned a data block as its memory (instance data block). The parameters that are

---

transferred to the FB and the static variables are saved in the instance DB, while temporary variables are saved in the local data stack.

The data saved in the instance DB are not lost when the execution of the FB is complete. However, the data saved in the local data stack are lost when the execution of the FB is completed.

An FB contains a program that is always executed when the FB is called by a different logic block. Function blocks make it much easier to program frequently occurring, complex functions.

An instance data block is assigned to every function block call that transfers parameters. By calling more than one instance of an FB, you can control more than one device with one FB. An FB for a motor type, can, for example, control various motors by using a different set of instance data for each different motor. The data for each motor (for example, speed, ramping, accumulated operating time etc.) can be saved in one or more instance DBs.

- A function (FC) is a logic block "without memory." Temporary variables belonging to the FC are saved in the local data stack. This data is then lost when the FC has been executed. To save data permanently, functions can also use shared data blocks.

Since an FC does not have any memory of its own, actual parameters for it must always be specified. Moreover, initial values for the local data of an FC cannot be assigned.

In function blocks (FB), a copy of the actual parameters in the instance DB is used when accessing the parameters. If an input parameter is not transferred or an output parameter is not write accessed when a FB is called, the older values still stored in the instance DB (Instance DB = memory of the FBs) will be used. Functions (FC) have no memory. Contrary to FBs, the assignment of formal parameters to these FCs is therefore not optional, but rather essentially. FC parameters are accessed via addresses (pointers to targets across area boundaries). When an address of the data area (data block) or a local variable of the calling block is used as an actual parameter, a copy of the actual parameter is saved temporarily to the local data area of the calling block for the transfer of the parameter.

The possibility to have internal variables dedicated only to one function block allows keeping a clean and simple project. This is a great difference with respect to the Schneider one. There, the variables were all set as global inside the sub-module making really complex its understanding.

About the blocks shown in Fig., they can be organised in two categories. The blocks that execute the motion control logic and the blocks that manage the data. In the first category, there are: Axes control actions, Format change management and Motion control logic. They perform the logic necessary for the execution of different operations. In the second one, there are the remaining blocks: Axes modules, Initialization, Input and Output management and

---

Alarm handling. They allow managing the data making them available for the previous category or for the other parts of the project control algorithm, either the PLC or the above layer MM\_Machine.

### 4.3 Axis module

The first element present in both the SM is the Axes modules. Depending on the number of axes that it manages, an equal number of FBs is used in the initial networks of the modules. In this sense, the SM\_Layer is characterized by an FB for the master and one for the Layer axis. The SM\_Stretch has five different FBs for Master of the module, Pusher axis, Film pull axis and the two axes related to the Crank. Being both real and virtual axes and also independent or dependent, the FBs are not the same calls but with different instances. Apart from a similar organization, they have different blocks inside them. To start, these blocks are coded using the ladder language. Having themselves FB calls inside, allows a more visual expression of the active or faulty parts. In the firsts networks the LaxisCtrl library blocks, explained in a previous chapter, are used. In particular, they allow activating the motions needed by directly raising an input and also permit to see the operative state by looking at the outputs. To exploits these possibilities from a different part of the code, a structure must be used. Indeed, inside each Axis module two UDTs (user data types) are defined: *Axis\_Ctrl* and *AxisStatusWord*. They come from a structure defined in the "PLC data types" folder of the project tree in relation to the inputs and outputs of the block used from the library. The first one has actually two alternatives. Indeed, as the structure is kept the same for all the FB generated for the different axes, the structure variable is repeated in each of them. Therefore, it is defined as *PosAxisCtrl* and *SyncAxisCammingCtrl*, namely the two types of axis control FBs took from the library. In addition, each *Axis\_Ctrl* structure is paired with the configuration structure *AxisConfig*. This one, differently, is an already defined data type from the library and simply added to the code. As mentioned in the section dedicated to the libraries, this variable allows managing all the parameters necessary for the motion blocks used by the library FBs. These parameters set the operative conditions for the motion action that is selected, such as position, velocity and acceleration. After this, the Axis module continues until the end with networks dedicated to the management of the errors. The details of the motion control error management will be covered later. In the code, only the faults activation and reset are defined. Therefore, only this is for now described. The outputs related to the axis error are checked. In particular, three different errors are present; Technology object error, Drive error, LaxisCtrl error. Obviously, the second one is only present for the real axes as the virtual ones do not have a dedicate piece of hardware. First of all, in case one of the errors named above is present, the generic error bit is raised. It is also an output value of the Axes module FB making it available for the rest of the sub-module. For example, it is used in the action selection FB for the managing of the axes when an error

---

occurs. The errors reset follows in the networks. It is organised in more steps.

1. Depending on the LaxisCtrl library management of the errors, a configuration bit must be modified in order to reset a technology object error. For this reason, the first step is to set the bit high in case the related variable which checks if the error occurred is raised.
2. Before the reset command is activated, in the second step, the output *resetActive* is controlled. Indeed, this avoids problems inside the same cycle of the OB1. This approach allows to activate the reset and then check if the reset has been performed only during the next FB call, namely in the successive cycle of the OB1. So, in this step, if the output of the library FB is high, the reset command is deactivated along with the bit of the previous step. Even in the case of the reset called not for a technology object error, this reset does not affect the configuration of the axis because it would set the same value to the bit, namely 0.
3. In the third step the reset command is activated. This depends on the presence of one of the errors mentioned above and the activation of the reset call from the HMI panel. Under these conditions, the bit is set high until the activation of the previous network resets it.

The three errors are generated through the use of custom FBs. If an error occurs at a technology object (e.g. approaching a hardware limit switch), a technology alarm is triggered and indicated. The impact of a technology alarm on the technology object is specified by the alarm reaction. Technology alarms are divided into three classes:

- Acknowledgeable warning  
The processing of Motion Control job is continued. The current motion of the axis can be influenced, e.g. by limiting the current dynamic values to the configured limit values.
- Alarm requiring acknowledgement  
Motion jobs are aborted in accordance with the alarm reaction. You must acknowledge the alarms in order to continue the execution of new jobs after eliminating the cause of the error.
- Fatal error  
Motion jobs are aborted in accordance with the alarm reaction.

To be able to use the technology object again after eliminating the cause of the error, you must restart the technology object.

In the case of the Technology object error detection, the block related is always enabled. This FB takes as inputs two important parameters directly linked to the TO analysed: *TO.ErrorDetail.Number* *TO.ErrorDetail.Reaction*. In particular, inside the block the detail reaction is first checked. Depending on the value displayed, if it is different from zero an error to be handled has

---

occurred. Consequently, the number would be analysed in order to select a value from a defined structure of strings where in every position is described the meaning of the fault occurred. After this, through a concatenation of strings, in an output variable of the block the number followed by description is stored. In case of a detail reaction different from zero, the output related to the presence of an error would be also raised, *TecnologyObject.xError*. This allows notifying in the Axes module FB the occurrence of a fault.

The LaxisCtrl error, instead, is linked to an error that occurred during a Motion Control instruction. This means that something has failed during the execution of one of the instructions controlling the motion blocks of the LAxisCtrls FB. Errors in Motion Control instructions are signaled using the output parameters of the FB itself "Error" and "ErrorID" or by the *LAxisCtrl.GetAxisStatusWord* error output. Under the following conditions, "Error" = TRUE and "ErrorID" = 16#8xxx are indicated for the Motion Control instruction:

- Illegal status of the technology object, which prevents the execution of the job.
- Illegal parameter assignment of the Motion Control instruction, which prevents the execution of the job.
- As a result of the alarm reaction for a technology object error.

In this case, the error activation FB is enabled when the error output of the library FB is active. After this, the block takes as input the diagnostic output structure of the FB itself by looking at the variables "Status" and "Sub-functionStatus". Respectively, they store the status of FB when error that occurred (namely the general error that the FB detects) and the ErrorID of called sub-function (namely the specified error of the motion block that failed its execution). The subsequent string conversions and concatenation of the two codes allow storing on output variables the information that are going to be displayed in the HMI. In parallel, the output *LAxisCtrlFB.xError* is raised for the error management.

The last error is the drive one. This management appears only in the Axis modules of the real axes, namely the Layer, the Pusher, the Film pull and the real Crank axes. This error control needs a more structured code. Indeed, the previous errors are linked to software parts of the project. Instead, in this case hardware access is required. In order to perform such actions, a dedicated FB, the *SinaParaS*, is required. This FB allows performing an hardware access in order to read or write the parameters of the hardware indicated. Moreover, it is an acyclic block. This means that a positive raise of the enabling input must be detected in order to be activated. From this, the following Fig. shows the code for its activation. Firstly, if the block is not running and the start is not present, the new rising edge is set. Otherwise, the enabling input is reset. These lines allow maintaining low the enabling input until the previous



operations are terminated. At this point, the mode management of the block is performed. In case the block is in reading mode (namely  $SinaMode = 0$ ) and has terminated its operations, the relative read elements are controlled. If some errors are detected in one of the two variables  $SinaRead_1$  or  $SinaRead_2$ , the writing procedure is enabled. This occurs only in case the reset command is in the first place activated and the FB is not in a busy condition. In this way, a new cycle of the FB can be activated. Indeed,  $SinaOperating$  is reset and the start command can be set to one. After the termination of the error reset ( $SinaMode = 1$  AND  $SinaDone$ ), the cycle is restarted from the reading stage.

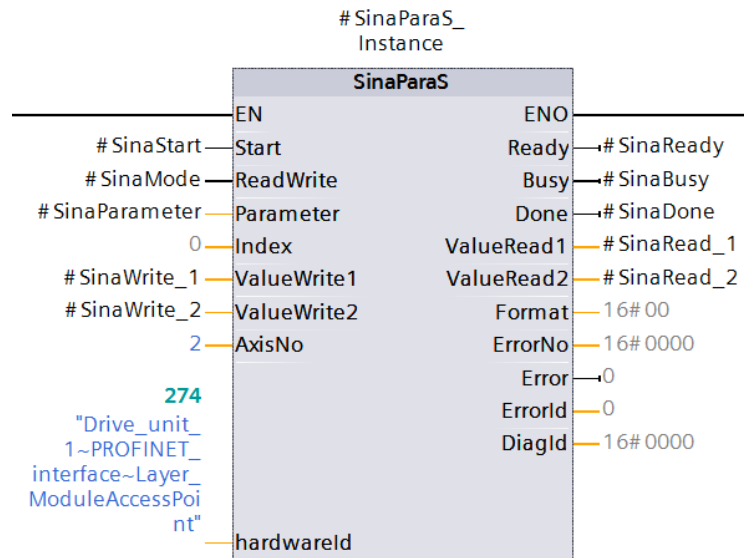


Figure 4.3: Drive error management.

The parameters that the block reads and writes are two specific values of the drive hardware, namely  $NumberOfActiveDriveAlarms = 2111$  and  $DriveAlarmAcknowledge = 3981$ . These two are found by looking at the parameters section of the motor. They are respectively linked to the "Alarm counter" and to the "Acknowledge drive object faults". To these, it is important to add also the two FB inputs displayed in Fig. before "AxisNo" and "hardwareId". As they were explained in the drive section before, they allow identifying the hardware we want to access.

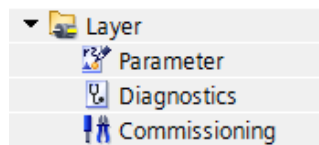


Figure 4.4: Axis Parameter.

---

```

IF NOT #SinaStart AND NOT #SinaOperating THEN
    #SinaStart := TRUE;
    #SinaOperating := TRUE;
ELSIF #SinaStart AND #SinaBusy THEN
    #SinaStart := FALSE;
END_IF;

// reading mode
IF #SinaMode = 0 AND #SinaDone THEN
    IF #SinaRead_1 <> 0 OR #SinaRead_2 <> 0 THEN
        #SinaMode := 1; // enable writing mode
        #StructError.Drive.xError := TRUE;
        #SinaParameter := #DriveAlarmAcknowledge;
    END_IF;

// writing mode
ELSIF #SinaMode = 1 AND #SinaDone THEN
    #SinaOperating := FALSE;
    #SinaMode := 0; // enable reading mode
    #StructError.Drive.xError := FALSE;
    #SinaParameter := #NumberOfActiveDriveAlarms;

// Activation of writing procedure to acknowledge the alarms
ELSIF #SinaMode = 1 AND #ResetCommand AND NOT #SinaBusy THEN
    #SinaWrite_1 := 1.0;
    #SinaWrite_2 := 1;
    #SinaOperating := FALSE;
END_IF;

```

Figure 4.5: Drive error management.

## 4.4 Initialization and blocks execution

An important aspect of the organization of the two sub-modules is how the different blocks in Fig. 4.2 on page 99 are executed. Two conditions are present: Axes successfully enabled and Initialization terminated. Both are necessary to be TRUE in order to activate the rest of the motion control FBs.

The first one is a requirement linked to the activation of the axes. As described in the Axes module part, in each of them there is a FB dedicated to axis motion management. However, even if the block is enabled during the first cycle of the OB1, the activation of the axis occurs only if a positive signal is detected in the "EnableAxis" input. Therefore, the activation of the block is performed only after some proper conditions related to the axis are present. This, indeed, requires the library described in the previous chapter, *IsAxisReady\_SyncAxis*. To perform the axes activation, multiples networks are used. In the first one, the library FBs are called. One for each axis of the sub-module, the one inserted must coincide with the typology of the axis to be controlled, either a PosAxis or a SyncAxisCamming. The inputs are the name of the axes to be controlled and a bit variable to select or not the check of the sensors of the motor. From these, the computation explained in the previous chapter are carried out, leading to a single output that notifies the possibility to enable the axis. In the code, this output is directly linked, through the

---

structure explained before, to the LAxisCtrl FB input that enables the axis. The results of the axes activations are used, in sequence, for the enabling of the sub-modules blocks.

The Initialization block condition comes after the termination of its execution. This programmable block is an FC. After its execution it is never used again until a complete reset of the machine. Therefore, no memory is required to be store in it. To perform its compilation only once, a first scan bit is set. Indeed, it is initialized at TRUE in the first call of the FB. Then, at the end of its first call it is reset to zero, leaving inaccessible the initialization block. Regarding its functionality, it is used to initialize some parameters that are going to be used throughout the code. In particular, it is divided into two parts:

- Configuration parameters

Inside this part, some configuration parameters of the LAxisCtrl library are set. Among them, the most important ones are those linked to homing of the motors, such as typology of homing, input sensor, direction, homing position and dynamic parameters. This aspect is very important because the homing procedure is more complex than Schneider's. Indeed, as the related section will explain more in detail, in this case Multiturn Absolute encoders are present on the motors. This results in a dedicated procedure for the absolute homing of the axes. Then, a second part is used in order to perform the re-phasing operations that can be required in some conditions. In addition to these parameters, some elements linked to the axes motion are present. They, however, are set at generic values to be initialized. Indeed, during the motion control steps, they are modified.

- Alarms definition

The second part is used in order to initialize the alarms of the sub-module. The details of the fault managing are reported later in this chapter. Here, the specifics of each alarm that is used are inserted. They consist of:

- type = this allows dividing the alarms for importance in different categories
- code = this gives an ordered list of the faults. It is used for the ordering also between the alarms of the same type.
- message = this is the detail of the specific alarms that occurred in order to have some information about it in the HMI display.
- reaction = this specifies for the motion control algorithm the procedure to be used in the case that alarm occurs

After this, the custom FB created for the fault handling is called. Indeed, this step is used as an initialization procedure also for that. This is the

---

reason why the alarms are inserted here. The details about these calls are reported in the Error handling section.

At the end of these steps, the status variable "Init\_Done" is set high. This is the last value missing for the activation of the sub-module blocks along with the enabling states of the axes. Indeed, they are set in sequence for the activation of the "Parallel\_Actions" variable. It enables the blocks making it possible to be executed by the CPU.

## 4.5 Sub-module bridges

The input and output bridges are used to match the PLC variables with the motion control ones. This allows working inside the algorithm with internal variables that are copies of the PLC values. These functions are built as FCs because there is no necessity for storing internal data. Indeed, on one side there are PLC variables stored in dedicated DBs, while on the other the motion parameters are stored in DBs dedicated to the sub-module. About the input bridge, it is divided into two sections. The first is focused on the simple copying of the variables, acting as a bridge. Among this class, there are the commands that the PLC passes to the motion such as the automatic or the manual modality of action. Moreover, there are some synchronization variables like the termination of the pushing phase. These, as the following description presents, are output variables of the module passed between each other. The second section contains some ladder logic. Therefore, it works as a filter where both the PLC and the motion parameters are combined for the activation of some variables dependently on the operative conditions. For example the conditions for the activation of the starting command of the layering axis.

With this FC, also another one is present. It is a FC dedicated to the management of the homing request and the motion enabling. The first requires the knowledge of the homing mechanism used for the sub-modules. Here, the logic executed in the SM\_Layer is briefly described. A homing request performed by the PLC is passed through the variable *x107.LayeringResetHome* in the first section of the the bridge. When this happens, the status of the axes is controlled. If the module is already homed and a homing procedure is not already running, the request is processed through the deactivation of the homing status and the enabling of the *iq.xHomeRequested* that notify its acceptance. Then, the latter bit starts the motion termination procedure, which is explained in details in the next section of this chapter related to the motion actions. These consequences are executed in the same OB cycle of the request acceptance. During the next one, if this procedure is completed successfully, the *iq.xHomeRequested* is deactivated in order to let the motion action automatically start the homing reset.

These same steps are also performed by the SM\_Stretch. The only difference is that in this case two types of homing reset are present. The first one affects the homing of the Crank axes and their master. Indeed, as the next section will describe, a dedicated homing procedure is present. The second one

---

is dedicated to resetting of all the axes of the sub-modules. For this reason, here two homing status bits are present: one for the first case  $xCrankHomed$  and the second for the other  $xCrankHomed$ . The layer module is only affected by the  $xOutputHomeOK$  bit that is set at the end of the homing procedure. For the SM\_Stretch, instead, it is set to TRUE at the end, but only when the other two are high as well. The module enabling part of the action FC is used for the deactivation/activation of the motion actions. It is set as follows:

$$xModuleEnable := xInputModuleEnable \text{ AND } xInputDelayedMainPower \text{ AND } xNewDataCycleDone \text{ AND } \text{NOT } FaultStopRequested$$

The variable  $xInputModuleEnable$  is a PLC bit passed through the bridge and is used in it in order to set from the control logic the enabling of the motors actions. The second variable  $xInputDelayedMainPower$  is a safety related bit. In this machine, the safety is managed with additional hardware dedicated to that functionality. Then, only if it send a positive status about safety, the motion is enabled. The  $xNewDataCycleDone$  is the status bit related to the format change. If it is TRUE, it means that the format parameters and the axes cams are successfully loaded, namely the machine can produce the desired bundle. The last variable,  $FaultStopRequested$ , is negated. It is linked to the custom faults management internal to the sub-module. This bit is activated only in case an error has occurred in some part of the machine. Therefore, this allows starting the stopping procedure of the motion algorithm.

## 4.6 Motion control actions

The core of the motion control logic resides in this block or, more specifically, in this group of blocks. Indeed, it contains multiple FBs that allows to select and execute the operation requested. This group characterizes both the sub-modules and, even if the structure is the same, the motion actions are different among the two. Obviously, this comes from the difference in the number of axes they have and the operational complexity that they must perform. To start, the common structure of the block is presented. As said, this is equal in both the SMs. This allows introducing, at first, the management of the actions and, then, to describe the specific computations and operations that they performs.

### 4.6.1 Motion block structure and management

The motion actions are structured as Fig. 4.6 on page 110 shows. The first FB to be executed in the networks sequence is the SelectAction. Here, the commands that come from the PLC are processed to enable the relative operation mode. Moreover, after the termination of one of them, forced or for the rising of a different command, this FB manages the stop and the ending of the active process. To perform such operations, this FB is characterized by different inputs. The most relevant ones are: the direct PLC commands that

---

are passed through the input bridge, the enabling bit of the motion actions FB (*iq\_xHome*, *iq\_xMan*, *iq\_xAuto*) and the faults reactions requests. This FB is coded with a ST case statement which allows having waiting moments and the execution of the command that characterize this specific block. Moreover, this allows its division in the two functionalities described above.

The first one is dedicated to the activation of the FBs. To perform such an operation, the first state processes the possibility for one of them to be active. This depends on the status of the motion algorithm and the commands of the PLC.

*i\_xModuleEnable AND NOT i\_xHomeRequested AND (i\_xInputRqMan OR i\_xInputRqAuto OR i\_xInputRqJog OR (i\_xInputHomeEnabling AND NOT i\_xOutputHomeOk))*

The *i\_xModuleEnable* is computed in the Input action described in the previous section. This allows establishing the circumstances that permit the motion to be performed or to continue. So, in the case of a FALSE state of this variable, the actions can not be activated. The same reason affects the specific action requests variables. To proceed in the SelectAction FB cases, indeed, at least one of them must be active. This affects also the homing starting procedure that, more specifically, can only be performed in case the axes are also not homed. For the other conditions, it is also related to the homing request, *i\_xHomeRequested*. It was also described in the previous section along with the module enabling bit. In particular, it is used for the termination of action but, here, it is necessary to not allow a new action to be executed until the activation cycle of the home request is terminated. In case an action can be activated, the case proceeds in order through the enabling states of the actions. This allows giving, by project choice, importance to the different conditions. Firstly, the homing state is checked. If a request is pending, the homing action is enabled through the bit *iq\_xHome*. Otherwise, the control passes to the manual state. Here, if its PLC command is active, the bit *iq\_xMan* is used to enable the correspondent FB action. The last case follows from a negative response to the previous where the automatic operation is activated with *iq\_xAuto* if requested. The possibility for one of the states to not be active is present. In that case, the FB returns to the initial state. Differently, if one of the states is active, the case switches to a new waiting state. Every time one of the motion action is active, this is the condition in which the SelectAction FB is. Here, the activation by the motion FBs of the variable *iq\_xSelectCase\_ExitAction* allows to continue to the second part. The logic for the activation of this variable is described later.

The second section of this FB is used to manage the axes after the termination of one of them. The axes are directly controlled by giving to the LAxisCtrl FBs instructions through the control structures defined. Depending on the sub-module, a different number and type of axes can be controlled. However, the logic performed is valid for both cases. The motion stop takes into consideration different possibilities. A PLC request to terminate comes

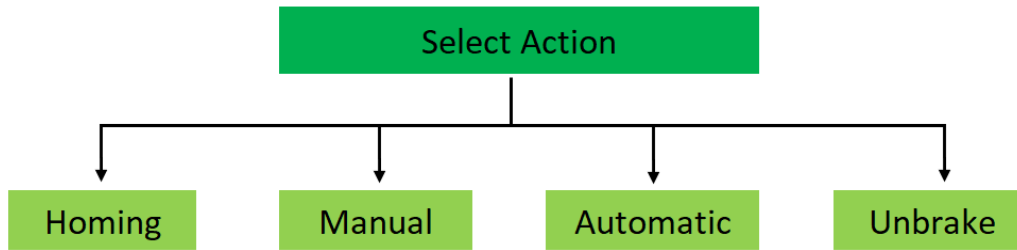


Figure 4.6: FB organization of the motion actions.

because a new operation needs to be executed or the one active is no more required. So, in both the SMs the master motion is controlled. Indeed, if it is moving, it is stopped and a positioning command is given to finish the operations in the starting angle, i.e. 0.0 degrees. Since the configuration parameters of the masters LAxisCtrl FBs could have been modified during the nominal operations, here they are re-defined in order to successfully completed the one required. After this or, eventually, if the axis was already not moving, the logic passes to the final steps of the procedure where the stop commands for all the axes in the sub-module are enabled to fully terminate any other possible pending command. This is performed to allow a safe return to the initial state of the FB by having the "standstill" status active in all the axes which leads to the FALSE value of the output variable *i\_xOutputAxesMooving*. In case of a fault termination request, one or more of the reactions (*xAutomatic\_Stop\_Request*, *xImmediate\_Stop\_Request*, *xPhase\_Stop\_Request*) provided by the alarm management explained in one of the section below can be active. Again as for the motion activation, the priority is provided by the order of processing used. From the less threatening one, the phase stop requires the completion of the active motion; namely, the command is terminated in case of the homing or the manual action active, or the operation cycle in case of the automatic action. For this reason, here the master is controlled again. It is allowed to terminate the positioning command in case of an Absolute or a Relative one. If it is moving at a constant speed, instead, that command is terminated by the call of a new positioning request in the zero angle. The final states describe before are then executed for the correct termination. The immediate and automatic stop, instead, requires the direct stop of every type of motion. However, they need a different initial logic due to their different nature. The latter is a fault provided by the error of the moving elements while the former is a specified condition. This implies that the automatic stop does not call a command for a faulty axis. As a matter of fact, the immediate stop calls directly the stop of all the axes in the SM and then the terminal states are executed. In the other case, the axis errors are controlled for everyone, leading to a stop call only for the active ones. A different ending state follows this scenario. Indeed, the check of the successful "standstill" status of the axes is performed only for those that received the stop command. Then, only the check of *i\_xOutputAxesMooving* is performed before returning to the initial state.

---

The homing procedure is different in respect to that used in the Schneider project. There, homing is performed as a reset and re-positioning of the axes, also performed at each start. This necessity comes from the hardware of the machine. The Schneider motors are characterized by a single turn absolute encoder. This means that the position of the axes is known only while the machine is running. Also, from this the software can not store the information retentively. Differently, the new Siemens actuators are composed of a multi-turn absolute encoder. The resulting absolute homing procedure is then performed only once. Consequently, the position defined as "home" is stored in the memory and saved. In this sense, to keep the re-phasing modality, a new piece of code is added.

The homing is managed in the TIA Portal in different ways. Homing is a requirement for displaying the correct position for the technology object and for absolute positioning. It can occur by means of an independent homing motion (active homing), the detection of a homing mark during a motion initiated on the user side (passive homing) or a direct position assignment. A distinction is made between the following types of homing:

- Active homing  
Active homing initiates a homing movement and performs the necessary homing mark approach. When the homing mark is detected, the actual position is set to the value specified in the retentive memory. It is possible to specify a home position offset.
- Passive homing  
The homing job does not perform its own homing motion. When the homing mark is detected during a motion initiated on the user side, the actual position is set to the value specified in the memory.
- Direct homing  
With the homing job, the actual position is set directly to the value specified in the memory or is offset by this value.
- Absolute encoder adjustment  
The absolute encoder adjustment compares the existing absolute actual value with the value specified in the configuration parameter of the homing block.

The direct and the absolute encoder are the only ones of interest. The former is the one necessary for the absolute homing of a multi-turn encoder. However, it needs the movement of the axis by the logic or by the operator. The latter allows storing the position indicated in the parameters as homing position to the axis. They are both implemented. In particular, in order to perform both homing and re-phasing, a dedicate FB for the homing is created. In this way, the logic created for the homing of the axes is kept, especially for the SM\_Stretch. This FB is divided in two different sections depending on the modality to be executed. The first state of the case is used for the



---

modality selection. This is performed through an integer that is equal to the number of the modality to be performed. In the case of the absolute homing of the actuators, the motors must be moved. Indeed, in this case the homing command is launched as direct only to store the actual position of the motor. Then, the motors are moved through a speed motion until the homing sensor is triggered. At this point, the true absolute homing is activated and the real position of the axis is stored. In case of a re-phasing motion, the axis is controlled through a positioning to the real homing position. This is necessary in some cases when the homing is used for specific purposes. These are explained later. This procedure allows you to emulate the homing movement while, in fact, it is not. Indeed, if a new homing of the axis is actually requested, the procedure activated by a specific input allows executing the previous algorithm.

### 4.6.2 SM\_Layer

The layering module is the simpler one for the number of axes and operations it must execute. As a result, the action FBs are less complex than the other case.

**Homing FB** Starting from the Home FB, it is used for the homing of the axes. By exploiting the procedure previously described, here it is only required the activation of the inputs. Indeed, in the first state the master and layer homing commands are activated. For the former, a normal direct homing is triggered. Being a virtual axis, the position at zero degrees is directly imposed. For the latter, the custom FB input is enabled. It executes autonomously the procedure until completion. Then, the last state modifies the homed axis variable to allow the start of all the motions.

**Manual FB** The manual operations are performed in order to execute the set-up of the machine. The functionalities and, therefore, the code can be divided into two main types of operations. However, the second is the execution in bypass mode of the nominal operation. It is also present, coded with the same logic, in the automatic action. In this sense it is explained with the rest of the "Auto" FB. In any case, its presence is required for the combination of the layering motion with the specific operations that the SM.Stretch executes in its manual action. Regarding the first part of the layer manual operation it affects only the layer axis. Namely, it is necessary to set up the mechanical elements of the machine. The layer axis is required to execute two different positioning operations: one to reach the lower part of the buffer (namely the first trap) and the second to reach the second trap that coincides with the bundle ready position. To perform these operations, absolute positionings performed by the master are exploited with the slave coupled through the cam. Firstly, in the case states the check of the bypass mode selection is performed. This allows leading the logic to the bypass states rather than to the manual positionings. In the latter case, the master is first returned to the

---

zero position. Then, the Layer axis must be linked to the master with the cam. For such a goal, the *MC.GETCAMFOLLOWINGVALUE* is exploited in order to get the position to be reached in relation to that of the virtual axis. The absolute positioning of the layer ends the initialization phase and starts the actual manual operation logic. Here, two variables are processed: *xInputManCorsaPrestr* and *xInputManCorsaStrat*. They are PLC commands filtered by the bridge that activates respectively the short and the long stroke of the layer axis. In particular, they are modified by the HMI panel where the operator can call off the positionings. When one of them is enabled the code assigns the related layer stroke to the cam input of the LAxisCtrl FB along with the position that the master must reach. Then, the cam coupling is enabled for the layer and the absolute positioning of the master follows. At the end of the motion, the machine waits for the command of the HMI selected to be deactivated. In this way, the machine returns to the initial state where the initialization is performed again.

**Automatic FB** The automatic action is the principal operation performed by the machine. Here, cooperation with the other sub-module is necessary. The code is organised again accordingly to the functionalities it has. They are performed after an initialization phase. To start, the layer axis is moved through an absolute motion in its starting positions, the lower possible one. This is not performed for the master because during these operations it is not used for a reference position like it is in the other SM. So, after this, the job selection takes place. In particular, like the previous action, it is divided into two operations performed: nominal and bypass modes. The first one allows the machine to work automatically without the necessity of the operator in order to produce the bundle required by the format. The second allows the operation of the machine linked to the SM.Stretch, performing only one layer per bundle and the push of the layer out without the film cutting and sealing. Therefore, the mode selection case is built around the value of the *i\_xInputByPassModeSelection* variable that allows switching to the bypass mode if active. In normal conditions the case proceeds with the nominal automatic operation. Firstly, the configuration parameters of the axes FBs are set. Regarding the master, the reference velocity is defined. This represents the speed of the virtual motor to which the layer axis is going to be attached through the cam. Indeed, the idea behind this build is to enable the motor cam when the packages are ready to push a layer. In this way, it is easy to manage the axis as only one is present in this sub-module. Instead, as the description of the next one will show, the management of the axes is different. Therefore, the layer axis configuration parameters defined are the profile reference and the cam mode that here are defined with the respective values of 2 and 0. They represent the direct coupling of the cam to the master independently to its value and the single cam execution to have the profile not be repeated when its end is reached. At this point, the start of the operation is waited. Before the real motion, the master must be activated. This occurs when the automatic command from the PLC is enabled and the start command is active.

---

The latter is the one that allows distinguishing between the different types of automatic cycles. One is the empty load run, where the machine runs without the presence of products. The other is the nominal operation. Their selection is also filtered by the input button on the HMI panel that allows the start of the operations. When the master is running at the reference speed, the layer axis is controlled. Here, the logic is used to chose between the short and the long cam strokes and it depends on different parameters. The former is selected when the machine can stuck multiple layers inside the buffer as the loading procedure. The latter, instead, is used only to empty the buffer while also adding the last layer. For this reason a dedicated logic is built. The start command, that enables the master motion, and the layer axis is ready to be used are the two main conditions that are necessary for both circumstances. The product-related conditions and logic are added to this. The short cam is firstly used when the number of layers stacked in the buffer (*G\_iOutputNrOfPreLayers*) is lower than their maximum number (*diInputBufferPreLayerNr* computed in the format change block) that it can contain. The first term is updated cyclically by the motion logic block described in a different section of this chapter. The short cam is used until the last layer is reached. In case the number of layers prepared already is larger then the maximum for the buffer, an alternative condition is used. Indeed, the synchronization with the rest of the machine must be active. Namely, the bundle is not completely formed and the pusher is in its back position. About the long cam, instead, it is used when the last layer is reached and the lasts conditions described of the short cam hold. This logic allows assigning to the axis input cam the one needed. Moreover, the values required for the counting of the layers are updated. In case the automatic command is reset while on this state of the case, a stop command is given to the master and the logic returns to the configuration parameters definition where, indeed, this variable must be TRUE. The continuous of the layer motion is composed of the direct activation of the cam command for the FB that is held until the synchronous status is reached. At this point, the operation is almost terminated. Indeed, as soon as the motion is completed thanks to the cam configuration parameters, the logic returns to the cam selection state where a new run can be performed.

**Bypass mode** The bypass mode is a modality that implies the use of the machine in order to produce only one layer of the bundle without the film operations. Indeed, during this mode, the Crank and the Film pull axes are idle while the film itself is removed from its standard position across the bundle motion path. To simplify such operation the two sub-modules works with a level of communication higher than in the previous scenario. This mode comes with a new project choice with respect to the Schneider software. There, the layer axis is directly attached to the SM.Stretch master. This is performed in the format change section. Indeed, in the MS260 machine, the enabling bit of the bypass mode is modified as a format parameter, requiring a new computation when it is changed. This type of operation can not be performed in the Siemens software because of the management of the LAxisCtrl FB.

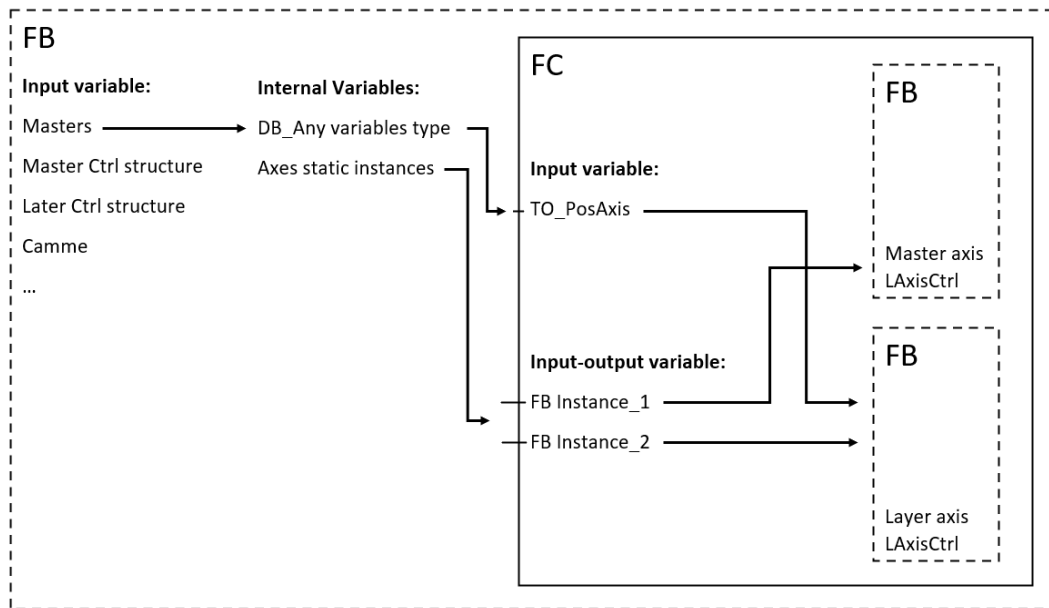


Figure 4.7: Bypass Solution.

Indeed, to perform such modification, a variable should be assigned in the master axis input of the desired axis, as it is done for the cam. However, among the variable types the `TO_PosAxis` allows to define an axis technology object but it does not allow to change the reference axis assigned to it, leading to an error during compilation. A solution that allows performing such an operation, namely to change the master axis of a slave, is to use a complex custom FB. Figure 4.7 shows how it should be structured. The organizations is divided among several functions and variables. The external FB requires as inputs all the variables necessary for the management of an axis, like the structure `AxisCtrl` defined in the previous sections or the cams linked to an axis. Then, the masters that need to lead the axis are also inserted and defined as `DB_Any` (general definition of a DB). Then, as internal variables other `DB_Any` should be defined along with the instances of the `LAxisCtrl` that contains the variables of the FB used to manage an axis. Therefore, the masters are passed through a specified logic to the input of the FC. It also takes as input-output variables the FB instances. These variables are, in this way, passed to the FBs. Especially, the master can be known as defined for the slave axis. This approach is too much articulated and rather useless for the linking of the layer axis to the `SM.Stretch` master.

An alternative could be the control of the layer linked to its standard master. This one is then controlled through a velocity set-point equal to that of the `SM.Stretch` master. However, the mutual interconnection of the sub-modules for this operative mode is lost requiring a more complex condition logic for the synchronization.

A simple and direct solution is used. The `SM.Layer` master is defined during its creation as a `SynchronousAxis` instead of a `PosAxis`. This allows exploiting the positioning features of the latter while also allowing to couple

---

it to the motion of another axis. So, the master of the layer is used as a slave itself in relation to another axis. Its master is, in this sense, defined as the virtual control axis of the SM.Stretch. This leads by commutation to the dependence of the layer to that axis.

Now, it is possible to explain the code used for the bypass mode execution. Once the initial states have defined the active condition of the bypass mode, the configuration parameters related to the acceleration and the velocity of the layer are defined. This allows for executing absolute positioning. The position is computed from the one specified by the actual degree of the master in the bypass cam. The master degrees are zero for the initial positioning imposed on it. Then, after its completion, the layer is controlled to be linked with such cam to the master. In this case, the cam configuration parameters mode and profile reference are defined 1 and 2; i.e. a direct coupling of the axis to the master and a cyclic execution of the cam to repeat it until a new command is sent. At this point the synchronization variables for the motion are checked. The master configuration position is defined with 130 degrees of phase shift with respect to the SM.Stretch master. This is required as the bypass cam provided consists of a profile for the pusher and one for the layer which is shifted of such quantity. Namely, the resulting positions of the axis are such that when the former is in its back position, the latter is in its extension phase. When the variables allow the continuation of the operations, namely the stretch sub-module axes are standstill and the start input is active, a positioning is called for the master. This drives the layer to its top position due to the phasing shift defined before. Then, a gearing coupling between the layer master and the master of the other sub-module is activated with a ratio of 1:1. In this way, the two masters start moving at the same speed, resulting in a fictitious coupling of the layer with the other master. This coupling is stopped when there is not a new start command from the product sensor. This implies that a new layer is no more available to be pushed. So, the gearing is interrupted by a new absolute positioning command for the master of the layer. This allows to end the current cycle of it and then stop the axis at zero degree which is when the layer is in its lower position. In the end, the logic returns to the state where the start is waited and the new position of the master is defined. This operation has an advantage. Apart from simplifying the master selection problem, the cam coupling of the layer is never broken. The only axis that is controlled is the master as the positioning and gearing commands are related to it. The axis to be controlled is only one through operations more simple and less problematic as a cam coupling.

---

### 4.6.3 SM\_Stretch

The stretch module is more complex for the number of axes and operations it must execute. As a result, the action FBs are more articulated than the other case.

**Home FB** Starting from the Home FB, it is used for the homing of the axes. By exploiting the code described in the MM\_Machine section, here only the management of the axes and the activation of the commands are required. Moreover, this block is more articulated than the SM\_Layering. Here, two parts are present: one dedicated to the pure homing, the other to the homing related action which is the sealer cleaning. The first is articulated in different steps. This SM is composed of multiple real axes among which the Crank is connected to a pneumatic actuator. Then, its motion implies the movement also of the other that is characterized only by a brake which must be disabled. In this sense, at the beginning the status of such actuator is controlled. A successful result allows enabling the Crank homing procedure through the custom FB developed dependently on the modality required. Then, the homing of the other axes is performed. Again, like in the SM\_Layering, the master is controlled through a position definition of zero degree. Then, the Crank is moved to the starting position and the homing status variables are modified. In the second part, three steps are repeated. Each of them consists of the deactivation of the pneumatic actuator linked to the Crank and the successive absolute positioning of the last. From this, the sealer can be cleaned and a new homing request is raised.

**Automatic FB** The automatic action is the principal operation performed by the machine. Here, cooperation with the other sub-module is necessary. One of the principal characteristics of this module is the cam management performed for the Crank and the Film pull axes. They have an A and B cam regarding the motion. This is a requirement for the online update of their trajectory which is explained in detail, later, in the motion logic block. There, the creation of the new cam is performed and the selection variable *ChangeCrankCam* is managed. The latter is set to TRUE only in case the cam stroke needs to be changed at run-time. In the code, the cam management is divided into two different parts. Outside of the case, the condition relative to the *ChangeCrankCam* is always checked. When it is active, the cam in input to the LAxisCtrl FB of the slave is modified. This does not affect the actual motion of the axis which, instead, continues to execute the old one. After this, the condition variable is reset along with the activation of the *WaitToChange\_CrankCam* or *WaitToChange\_FilmCam*. They are used in the second part of the related code. This is performed inside the case, especially inside the active states. They are those in which the code can be stuck for some time while waiting for a condition to be verified. Therefore, during that time much larger than that required for a transition state, the cam-change takes place. A particular output bit of the LAxisCtrl FB communicates if the

---

cam selected is active. In this way, the input that enables the cam coupling can be always reset if the output is high. From this, it is possible to give a new positive edge to that input in order to start a new coupling. This is used as a condition for the activation of the online defined cam instead of the one used at the beginning of the case. Indeed, the input that enables the use of the latter is reset right after the synchronization of the axes through it has occurred. The next condition for the online cam change is activated when the master reaches the phase stop position (that is at zero degrees when a new cycle starts), and the *WaitToChange\_xxx* is activated. When this happens, the "change" variable is reset and the command for the cam coupling is raised. This sequence of conditions is fundamental because in this way it is possible for the slave to follow the new cam at the beginning of the successive OB1 cycle. Lastly, the cam input is reset as before as soon as the old ones have been processed in order to be ready to repeat the logic if required.

In parallel, the motion algorithm is organized again accordingly to the functionalities it has: the nominal and the bypass modes. The former is used as main logic. It allows the creation of bundles in synchronization with the SM\_Layering. The initialization performs a set-up of the axes conditions to allow a good start. In this sense, the Crank must be specifically controlled. One of the first states is dedicated to the configuration of the cam commands. Mode, profile reference and positions are defined. In particular, the simpler case defines a direct and cyclic cam for the Pusher and the Film pull and the latter also for the Crank. In the case of the Crank axes, different scenarios are possible. The bypass mode sets the axis in an idle position that is calculated during the format change. It coincides with the highest position of the cam profile in order to allow the bundle ready to pass. The master in the starting position allows initializing the axis in the higher state. This is mostly performed after the homing procedure or at the end of a cycle where the Crank is in the lower position on the base of the cam profile. Then, the master can be in a different position allowing to directly activate the cam coupling. After the configuration, the axes are positioned accordingly to the master angle through the *MC\_GETCAMFOLLOWINGVALUE* FB. The one related to the Crank is only executed in case one of the previous actions commands it. Otherwise, the positioning is performed to the upper state. Lastly, the initialization terminates with the cam activation for all the axes. Also the Crank is managed here but, in one circumstance, the cam is followed only from a specific master degree. The proper control algorithm is executed afterwards. When the automatic conditions arise, the master is activated to execute one cycle. This considers both cases where it is at zero degrees or in a different position. In the latter, the master follows a positioning command to reach the starting angle and then, as in the former case, the automatic cycle state is reached. The starting inputs are related to the correct completion of the bundle by the first sub-module and the start from the PLC is available. This allows you to first set the speed parameters and then to give the input command. The operations are then terminated when a new input does not arrive, especially from the ready bundle, allowing to perform an absolute positioning which cancels

---

the previous constant speed command so that the cycle currently in progress can be terminated.

The bypass mode is performed in the second part of the case. This works in direct connection with the layering module. At first, the Pusher is linked to the new cam to be tracked. Then, the Crank axes are positioned at their idle states. The last states are used for the motion. As the previous part of the bypass in the SM\_Layering described, the coupling is essential. Here, indeed, the master velocity parameters are defined waiting for the synchronization with the other axes. This allows the start of the master. It drives also the Layer coupled with its master that operates in gearing 1:1 with the current master. As in the nominal conditions, the cycles continues until the Layer stops, imposing a stopping to speed with a positioning. Then, the system returns to its waiting position to restart the cycle.

**Manual FB** The manual operations are performed in order to execute the set-up of the machine. The functionalities and, therefore, the code can be divided into two main types of operations: the jog motion and the welding cycle. The first one is related to the activation and control of the axes for the mechanical set up of the machine. This operation is performed through a "Jog" motion of the axes. This means that the motion is directly controlled by the operator which moves them through small displacements through the trigger of a panel button. This modality merges both the operative conditions of the layering module, namely the nominal and the bypass mode. Both are performed under these circumstances, but still taking into account their differences. Below, the former one is described, but the passages are used also for the latter with the difference that its peculiarities are applied. This code contains the same axes initialization that is used for the automatic action and, therefore, there it is explained. For completeness, the initialization is dedicated to the selection of the Crank operative condition, the positioning and the cam activation of the axes. About the latter, the management of the A and B cams, like for the Film pull axis, is necessary. Again, this is used likewise in the automatic FB. After the start-up, the master axis is taken into account. The positioning configuration parameters (position/distance, acceleration, deceleration) are defined. They affect both the absolute and the relative motions. Indeed, in this state the Jog activation is processed and, in the case of positive input, the axis is moved. This comes under two circumstances. In one case, the master is already at zero degrees making a new absolute motion in that position impossible. Therefore, a 360 degrees relative positioning is used. This allows completing a full cycle if it is allowed. In case the master is not in its starting position, the absolute motion is performed with zero degrees as the target. Once one of the two commands is activated by the input sent by the operator, the logic waits for the release of the Jog button. This implies the enable of the stop command for the master. Then, the case returns to the master activation state where one motion between the absolute or the relative is activated. This code logic allows performing only one operative cycle at a time. Indeed, when the master returns in the starting position, the axes are standstill waiting for the release



---

of the button. This meets the project choice made for the Schneider version of the MS260. A different solution would have been the use of the Jog modality implemented in the LAxisCtrl FB that controls the axes. In this circumstance, the input signal could have been directly connected to the FB input making the control easier. However, the direct connection would have made the managing of the axis directly dependent on external input from the case logic. Therefore, to compensate for this, the implementation of the "one cycle at time" characteristic would have been less intuitive. The solution adopted comes from the necessity of better adapt the Schneider version. There, the start and stop motion is velocity-dependent through a condition external to the case. In it, the relation to the activation of the input from the operator, the axis velocity is set at a reference value or at zero. Therefore, the motion is made through the modification of the speed that allows to stop or move the axis. This approach does not exploit the features of the motion logic, making mechanical a rather simple motion.

The welding cycle is the motion in the second part of the manual case. It is implemented to perform a cycle with the Crank axis without the Pusher and the Film pull. For the sake of mechanical set-up, this allows to directly control that axis. The implementation is performed by adding a jump condition in the state for the master configuration described before. If the welding request is active, the case passes to the relative states. Moreover, an external condition to the case set the reference speed of the master for this motion. This is used also for the Jog mode in order to set the velocity of both the absolute and the relative commands. In this modality, differently from the previous, the master is only controlled through a relative motion. Indeed, the cycle is performed entirely without stops. Therefore, the same conditions evaluated previously apply again as it starts from zero degrees. The initial states of this mode affect teal axes management. For the two not used in this mode, they are stopped and the "idle" cam is activated for the Pusher. This is not a mandatory condition as the positioning performed before it is sufficient. The cam does not impose any motion. The Crank, instead, executes its cam. This is valid for both of the technology objects. Then, the states activate immediately the master without leaving any waiting state between the activation and the end. Finally, the case returns to the beginning.

**Un-Brake FB** The last motion action present end executed in the SM\_Stretch is the brake release. In the second chapter where the actuators have been discussed, there was described how the Crank motor must be provided of a brake. This is required for the vertical motion of the mechanism attached to it. Therefore, it is fundamental to have a brake when the motor is not in torque. This condition could be modified by an external request of releasing the brake. To do so, the PLC command drives the "Select action" FB to the activation of the "Unbrake" FB. This one allows enabling the request to release the brake from the Crank motor. Depending on how the hardware is managed in the TIA Portal, this control cannot be performed normally inside the motion logic. Instead, direct access to the drive parameters must be executed. To do so, a

---

specific FB of the library LAxisCtrl called "Brake Control" allows giving as input the technology object and the Execute command in order to access the drive parameter that controls the brake. This FB must be specifically used inside the MC-PostServo, a specific OB that is called at the end of the MC-servo execution. In this, the FB performs the command giving feedback on the success of the operation in order to exit from the Un-brake FB and to return the control to the Select action.

## 4.7 Format change management

The format change is performed in both the sub-modules in a dedicated FB, *SizeChange\_Layer* in one case and *SizeChange\_Stretch* in the other. The activation of the block execution is performed in the same way. The starting point comes from the MM\_Machine module previously described. As was mentioned before, the dimensions of the packages and bundles are computed. The parameters are passed to both the sub-modules through the common data block *stModulesInOut*. The MM\_Machine does not provide only the dimensions but also stabilizes the start of the dedicated format change computations. Fig. 4.8 on the following page shows how this is performed. In particular, the variables *NewDataOnline* and *NewDataCycle* are used. The same logic structure is exploited in all the modules for their application. The first one communicates the start and the end of the MM\_Machine format change. This variable is always TRUE when the computation in this module is not running; namely, there is no request for format change from the HMI. It is set to FALSE only in case a new one has arrived. This change of values allows in the initial states of the sub-modules Format change FB to recognise that a format change is in action. There, depending on *NewDataOnline*, the variable *NewDataCycle* is managed. As will be shown, it is used the enable the motion action. Namely, if this variable is not TRUE, it means that format parameters are not available and computation is required by the PLC. Therefore, the format change block waits in its first state until a FALSE is detected from *NewDataOnline*. This event triggers the modification to FALSE as well of *NewDataCycle* that coincides with the ending procedure of the axes motion. Then, when a new positive value is sent from the MM\_Machine, the computations of the sub-modules instructions is performed. This ends with the activation by them of the *NewDataCycle* that allows a restart of the axes operations.

The computations performed by the sub-modules are described separately because, even if it performs similar jobs, the conditions in which they work are different.

### 4.7.1 Layer sub-module format change

The states of the format change after the initialization are dedicated to the computations of the new cams. This sub-module is characterized by only one axis that is moving with a cam coupling, namely the Layer. It is paired with the master of this sub-module and here its cams are defined. The first part of

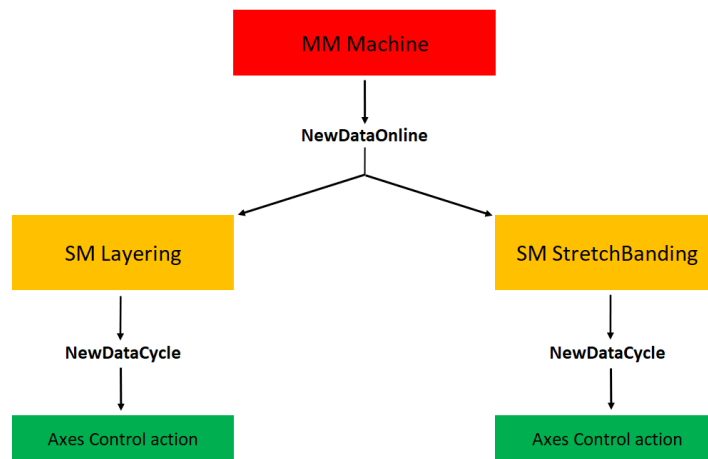


Figure 4.8: Format Change Organization.

the machine is aimed at the stacking of the product in layers for the preparation of the bundles. To do so, a buffer is placed between the two machine sections, allowing to stuck as many layers as possible before interfering with the second part. In this sense, the first instructions are aimed at the definition, on the base of the new packages and bundles, of the relations between the products and the machine buffer dimensions. Indeed, the geometry parameters that compose the buffer are taken from the system values stored in the PLC (such as the buffer high and the distance between the waiting products and the buffer) along with the new data that were not used by the MM\_Machine as it is for the product margin. With these parameters, it is possible to compute the number of layers that can be stored inside the buffer. Accordingly to the number of layers per bundle, the maximum number of layers can be stored until the number that composes the bundle with one less. In this way, the PLC sub-module knows how many travels it must perform before pushing the last layer along with those stored in the buffer to the bundle waiting position. Fig. 4.9 on the next page shows such conditions.

At this point, the cam positions are defined. Only two are necessary for this axis: the first trap position and the second trap position. The former is used to load the buffer, the latter to push the whole bundle. As a consequence, only trap distances are necessary for this job. In parallel, also the master positions are defined. In this case, they are kept from the PLC variables as the input of the format change block. The axis, then, is characterized by two main cams with 4 points each: the short cam and the long cam. They allow the operations just described. An additional one is present that is used for the special operative condition called bypass. It is similar to the long cam as the axis must push the layer of packages to the second trap. The only difference relies on the angles of the master that is paired to the axis. To create the cams the LCamHdl library is exploited. The properties of such block are explained in details in the previous chapter in the section dedicated to the libraries. Briefly, the cam points are defined with the advanced elements of the FB which are composed of the starting and ending position of a segment where the type of

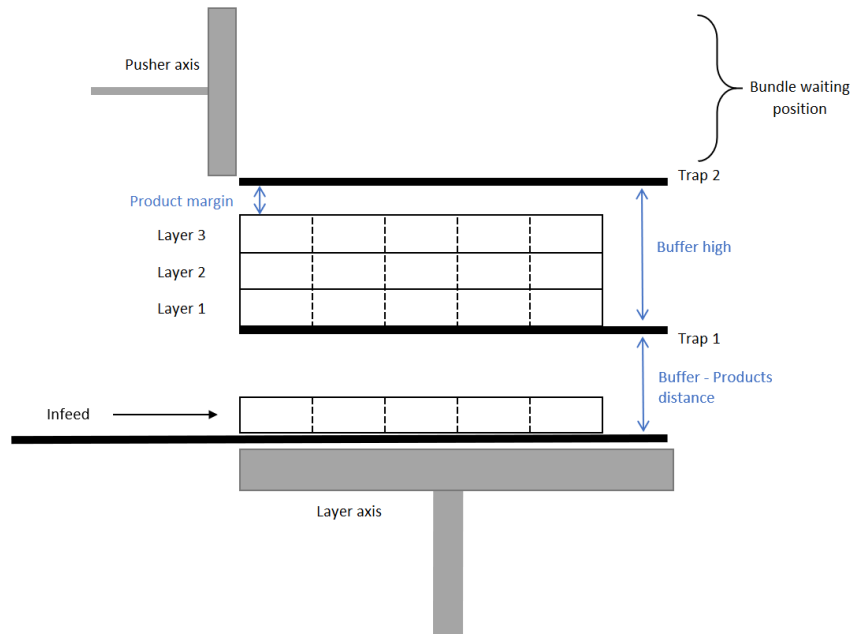


Figure 4.9: Buffer geometry.

segment is specified. The details of the segments are reported in the table below.

Short cam	Segment 1	Polynomial of 5 <sup>th</sup> order
	Segment 2	Polynomial of 5 <sup>th</sup> order
	Segment 3	Constant function
Long cam	Segment 1	Basic sine function
	Segment 2	Polynomial of 5 <sup>th</sup> order
	Segment 3	Constant function
Bypass cam	Segment 1	Polynomial of 5 <sup>th</sup> order
	Segment 2	Polynomial of 5 <sup>th</sup> order
	Segment 3	Constant function

The last step is the interpolation of the points. The array of elements just defined and the TO of the cam that must be interpolated with these characteristics are passed as inputs to the FB *CreateCamAdvanced*. The enable to the FBs is activated only when all the previous instructions are terminated. Then, at the end of all the interpolations, the cams in Fig. 4.10 on the following page are ready. As it is possible to notice, they are similar to those shown in the tests presented in the second chapter. Lastly, the status variable *NewDataCycle* is set to TRUE and the case returns to its initial state waiting for a new cycle. The FB calls are inserted at the end of the case statement that composes the format change routine.

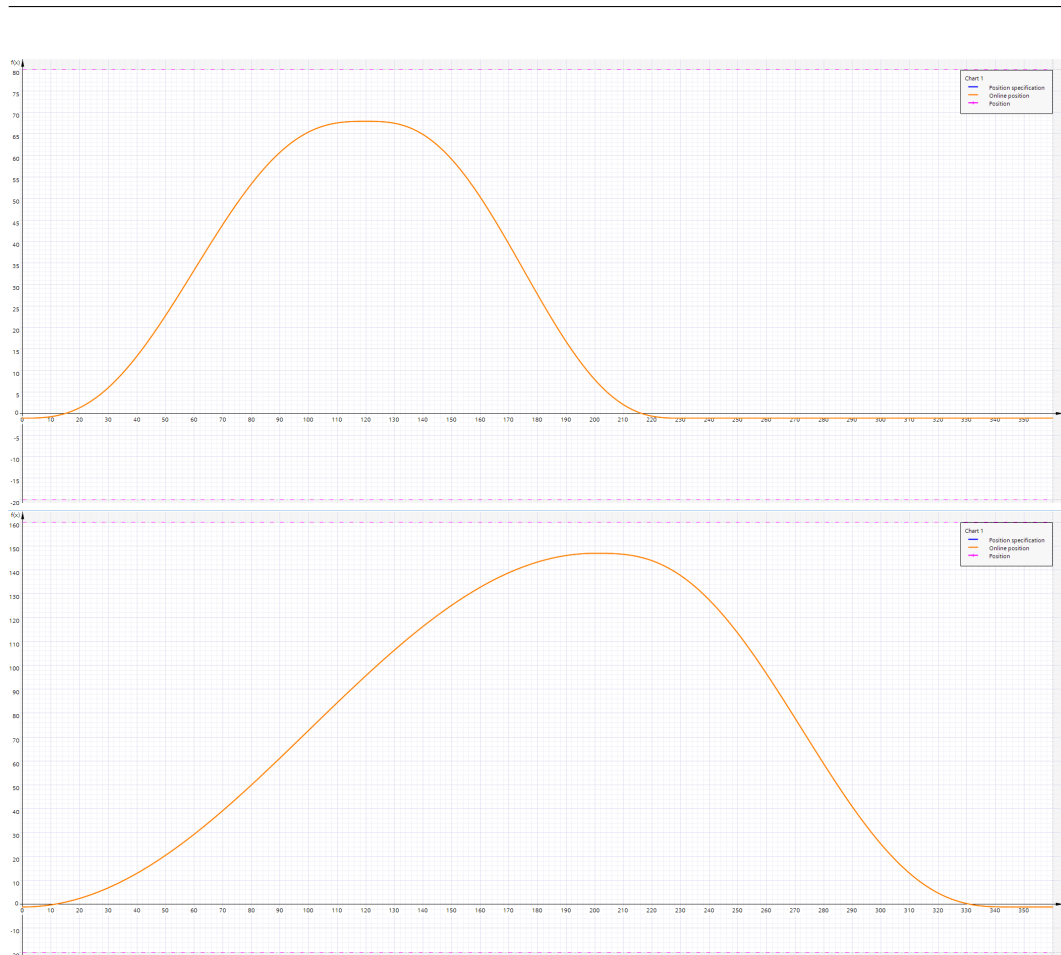


Figure 4.10: Layer axis cams.

In addition to these instructions, an "if" statement is present outside the case. This is because its instructions must be computed at every OB1 cycle. It is added to the block of the sub-module because it is strictly linked to the cams here defined. This piece of code is linked to the functionality of the HMI added to this project to show on the panel, when requested, the cams running on the motion control. The details of such code are described in the HMI chapter because it is not a fundamental functionality of the motion control algorithm.

#### 4.7.2 Stretch sub-module format change

The format change in this sub-module is more articulated than the previous. This is a natural consequence of the higher number of axes present and moved by a cam; three in this case: Pusher, Film pull, Crank. Again the logic is organised with a case.

**Pusher** About the first axis, it is managed as the Layer. Three cams are used for it: one for the normal operating conditions, one for the bypass modality and one for an idle state when the axis must stay still. At the beginning, the operative conditions regarding the new bundles are defined (on the base of

---

two margins, one for the length and one for the high of the packages). In this case, the resulting dimensions lead to four possibilities for the pusher motion of the nominal operative state. For this porpoise, the high and length of the bundle are used to identify if they are greater or less than a standard margin related to them. This, indeed, leads to four scenarios:

- package high  $\leq$  HighMargin AND package length  $\leq$  LengthMargin
- package high  $\leq$  HighMargin AND package length  $>$  LengthMargin
- package high  $>$  HighMargin AND package length  $\leq$  LengthMargin
- package high  $>$  HighMargin AND package length  $>$  LengthMargin

The differences rely on the master positions at which the pusher ones are paired. About the positions computed, instead, they are left unchanged and they affect: back position (where the axis waits for the bundle from the layer) forward position (where the bundle is left in its sealing position). The definition is performed inside a case in which the state to be executed is selected by the computation above. The other two cams are instead univocally defined from the pusher positions. To define the cams, again the *CreateCamAdvanced* FB is used. Then, the segments are defined as for the Layer axis and the table below shows their definitions.

Nominal cam	Segment 1	Modified sine function
	Segment 2	Polynomial of 5 <sup>th</sup> order
	Segment 3	Constant function
Bypass cam	Segment 1	Basic sine function
	Segment 2	Constant velocity function
	Segment 3	Polynomial of 5 <sup>th</sup> order
	Segment 4	Polynomial of 5 <sup>th</sup> order
	Segment 5	Constant function
Idle cam	Segment 1	Constant function

The interpolation is executed immediately after the definition of the segments. A dedicate FB for each cam is present in order to speed up the computations. So, as soon as they are terminated, the profiles shown in Fig. 4.11 on the next page are loaded and the format change block continues with the computation of the Crank axis.



Figure 4.11: Pusher axis cam.

**Crank** The Crank is the most articulated axis of the machine. This is the result of its mechanical composition that affects also the cam definitions. Differently from the Schneider software, the automatic management of the conversion between the linear motion of the Crank and the rotating one of the motor attached is not present. This requires the use of the custom library developed *Crank\_Module*. It is described in the previous chapter among the other libraries and in this block, it is exploited. In this case, two axes are managed: the real and the virtual. The former is related to the rotating motion of the motor, the latter to the linear axis in the machine. A cam dedicated to each one is present but, differently from the previous axes, they are only affected by one type of motion. To start, the positions of interest of the linear motion are computed thanks to the bundle dimensions and the static PLC parameters. They are specific for the three positions that the Crank follows. The traces in the second chapter allows us to appreciate them. The initial position is the "top" one. It is reached when the bundle is pushed by the first axis of the module. Then, the Crank gets down until the "low" position is reached. Here, the sealing and the cut of the film are performed. This action represents the central functionality of the machine. Indeed, specific parameters can be defined in the HMI in order to design the sealing phase as desired. Moreover, as the successive code computes, from this phase other positions are derived in order to build the motion of the machine accordingly to this core functionality. The last step is the "blow" position. After the sealing, the Crank moves up the sufficient space in order to retreat the blade. Here, the machines blow air in the sealing spot to cool down the sealed film. This operation requires a time specified from a format variable. So, this continues also at the beginning of the next cam cycle when later the Crank start to return to the "top" position. After this part, the cam segments are defined inside a case. As for the Pusher axis, this one is divided in the four alternatives described above depending on the bundle dimensions. To define the segments an array of custom structures is used. The relative reasons are explained in the *Crank\_Module* library section since in this case the standard segments can not be used. At this point, before terminating this part some additional instructions are performed. The

first is the computation of the master position relative to the Pusher one. It is performed through the Siemens FB *MC\_GETCAMLEADINGVALUE*. By specifying the following position of the slave axis, the block allows determining the first master angle that coincides with that in the cam provided as input. This operation leads to the exact moment in which the Crank should be in its upper position allowing the bundle to reach the sealing position. This is performed relative to the Crank position description above. The sealing and the successive blowing phases are the central functionality of the operations. Therefore, it is allowed for the Crank to remain in those phases for all the time possible. This leads to the computation of the Pusher position counting also the bundle dimension and some safe space in order to find to master degrees in which the sealer must leave the "blow" position. Then, the initial master angles of the Crank segments can be changed accordingly to the result. If the master position found is later than the one defined, the points are moved. To this, the computation of the rotating cam is performed by the custom library and the two cams are interpolated. Figures 4.12 and 4.13 on the next page show the two cams defined with this logic. At this point, some other positions are requested to be defined through the FB *MC\_GETCAMLEADINGVALUE*. The first one detects the exact master angle when the Crank moving from the "top" to the "low" position passes through the "blow" position. This is used later for the Film pull axis. Indeed, this position coincides with the instant in which the film must be pulled at its maximum in order to tighten the bundle. So, the master degree found is used later for the Film pull cam definition. Then, the Crank master angles are managed to find the midpoint of one of the segments, namely when it moves from the "low" to the "blow" position. This is then used to trigger the *outputCam* related to the blower of the machine. In addition to this, also other angles related to the Crank rising and before rising output cams are found. Lastly, the custom library FBs are exploited in order to define the important positions for the rotating axis that are used in the motion control block for positionings.



Figure 4.12: Crank axis cam.



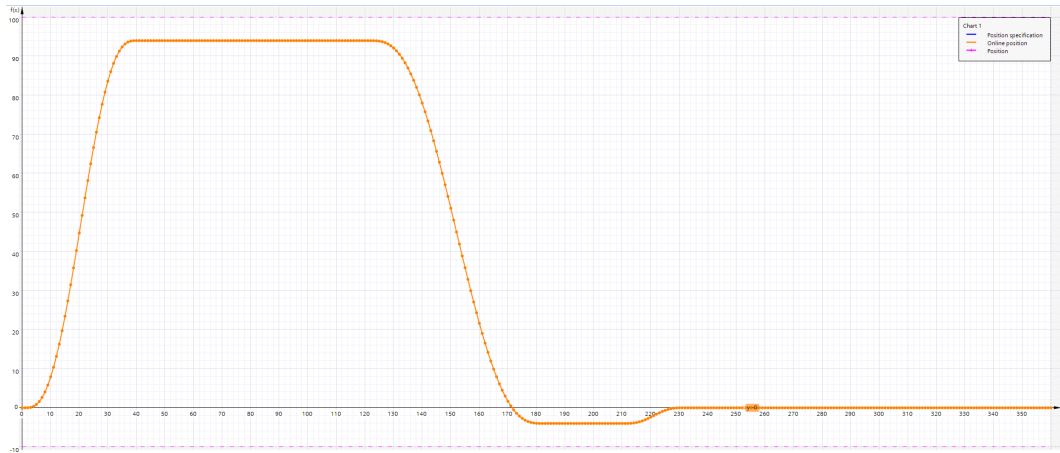


Figure 4.13: Virtual Crank axis cam.

**Film pull** The simpler format change is relative to the Film pull axis. The points of its cams are directly computed from the data coming from the PLC and its motion is exclusively controlled with one type of cam. Its role is to pull the film and this operation leaves the axes unused if not requested. The points of interest in this case are the starting angle and the maximum rotation to be reached. The latter is defined in the previous part of the block. The former is a static value defined by project choice. The interpolation is executed like for the Layer and the Pusher. The array of advanced segments is defined and then passed as input to the dedicated *CreateCamAdvanced* FB where the film pull cam is also specified. So, the resulting cam is loaded in the PLC as Fig. 4.14 shows it.



Figure 4.14: Film pull axis cam.

---

## 4.8 Motion control logic

The PLC logic block in the two sub-modules is a piece of code dedicated to additional computations that need to be carried out in parallel to the motion control algorithm that is executed in the block previously described. In particular, inside the structure of the module, this block is called after the motion algorithm that manages the axes. This is because after the motion of the axes the control variables are updated and the outputs are generated. These blocks perform specific instructions dedicated to their operations.

**Layering module** In the SM\_Layering, the motion logic is mainly aimed at the computation of the bundle layers produced by the axis. This is divided into logic for the bypass and logic for the nominal conditions. Indeed, the former mode requires only one layer to be pushed and so different computations. For the latter case, the instructions allow to manage, on the base of the format change parameters computed in first place, the variables *G\_iOutputNrOfPreLayers* or *G\_iOutputNrOfLayers*. These integers show the number of layers that have been added during one cycle of the machine to the buffer and to the bundle waiting position. This works in parallel to the motion block. The specific logic used by the motion algorithm is treated in the dedicated section. To be mentioned is that, in parallel to this logic, on the base of the cam executed by the Layer axis, short or long stroke, two variables are set: *iActualPreLayer* and *iActualLayer*. They are integers variables that are used to update the two number of layers mentioned above. A dedicated network in the output filter combines the total number of layers executed in the variable *iLayersDone* to match the maximum number needed. This, indeed, activates a reset for both the *G\_iOutputNrOfPreLayers* and the *G\_iOutputNrOfLayers* which allows a new counting for the successive bundle. These logic instructions are paired with some triggers and timers. Such functions are exploited in order to detect alarms or to enable the reset of the above variables under certain operative status.

**Stretch module** In the SM\_Stretch, the logic is dedicated mainly to the parallel work of the motion control linked to the cams. This block is not used for the help of the logic of motion control. Instead, the axes positions and the active cams are controlled. Starting from the former, this block is filled with function calls that use the output cams stored and defined in the axis MCV\_Master described in the technology objects section of the previous chapter. In particular, the ones used are:

- *SealerRising* This is the output cam dedicated to the detection of the master degree when the Crank axis is in its rising phase, namely between the position of sealing and the top position of the cam when the bundle passes underneath.

- 
- *Sealer\_Before\_Rising* In addition to the previous output cam, this is used for the detection of the Crank when it is not in the zero position.
  - *PhaseStop* In contradiction to what the name suggests, this output cam is used for the detection of the starting point of the Crank motion, namely when it is in the stop angles. This indeed is used for the check of scenarios of the motion algorithm when the stop of Crank is requested and waited.
  - *PressBulkPhase* This is one of the output cam used for the control of an external motor not defined through the Technology Objects. As the initial descriptions of the machines said, several pneumatic actuators are used in the second part of the machine. One of them is dedicated to the press of the bundle when it reaches the sealing position. This operation is necessary because, as it was described in the first chapter, the film must be pulled to have a firm bundle as output. Then, if a blocker is not used, the Film pull would bring the bundle out of its correct position. As a result, this output cam detects when the virtual master reaches the interval between the positions defined among the PLC system parameters. In addition to the other functions, ON and OFF compensation times are added in input. This is necessary because the pneumatic actuator cannot reach the dynamic performances of an electric motor. Then, to match perfectly the angles of the master to the activation and deactivation instants of the pneumatic actuator, compensation timers are added.
  - *Pusher\_FRW\_Phase* This is the only output cam used in relation to the motion of the Pusher axis. it is used for the detection of the pusher extending motion. Namely, when the master is in the angles that define the motion of the pusher between the back and the extended position, a bit is raised in order to let it know to the PLC through a status bit.
  - *LowerFilmPhase* analogously to the previous output cam, this is the only one dedicated to the Film pull axis control. Moreover, the detection is dedicated to the phase of the axis when it pulls the film between the master angles related to the rest and the maximum rotation of the axis.
  - *WelderBlow* This is the output cam dedicated to the blow of the cooling air in the sealing spot of the machine. This is activated thanks to the master angle, computed in the format change of this sub-module, related to the reaching of the Crank axis of the zero position after it has raised to let the bundle pass. This is only applied to the triggering of a variable. Then, it is used inside an OFF Timer where the time is taken again from a system PLC variable.

The second part of this block is dedicated to the management of the cams. The Crank axes and the Film pull one is characterized by two cams each, A and B. This comes from the necessity to change at run-time the cam followed by the axis. The Schneider cam management allows doing such operations

---

automatically. Indeed, a request of this type is performed by modifying the specifications of the points of the cam that must be different and adding a request of *NewCam*. Then, the code provides the execution of the new cam when the old one is terminated. This functionality is not present in the Siemens software but must be kept as availability to the machine logic. Indeed, the cam used by the axis can not be modified and a different one must be used. For this reason, an handling routine is added inside this block.

Starting from the Film pull axis, the fundamental cam parameters (such as *lrLowerFilmPullOn* that represents the starting angle of rotation of the axis) are at each OB1 cycle updated from their definition variables in the HMI. This allows the operator to modify them while the machine is running and to find the perfect ones. This implies a re-building of the cam used. When a parameter is modified with respect to the value of the previous cycle, the array of advanced segments used for the cam definition are computed again. This also triggers the modification call of the new cam. Fig. 4.15 on the next page shows the code implemented. Firstly, in such a condition, the cam to be modified must be selected. If the axis is not in "synchronous state" then the A cam is passed to the interpolating FB by default. Contrary, the one not used is selected among the two and the bit which communicates the necessity of a cam change is raised. The second state activates the interpolating FB. In addition, on the base of the bit, the module variable is activated if a change is requested or the A cam is assigned by default to the cam entrance of the LAxisCtrl FB that control the axis.

The logic for the Crank axes follows the same steps. In this case, the activation variable for the new interpolation is the degrees by which the axis must remain in the weld position. This is controlled by the welding time and the machine operative speed. Both of them can be modified from the HMI panel. After the new definition of the cam segments, a similar case of Fig. 4.15 on the following page is executed. The cam management and interpolation are executed like the previous one as also for the output variables of the last state for the cam execution in the motion control. Only a few differences are present. The first one is that the cam selection implies the definition of both the linear and the rotating cams. Then, the custom "*Poly5\_360\_crank*" FB for the conversion of the cam points for the rotating cam is executed. Lastly, the terminating state of the case must interpolate the new points for two cams instead of one.

This piece of code, as described, provides only the management of the new cam on the base of the active one. Indeed, the online cam change logic is directly added in the motion algorithm and it is described in the dedicated section.

```

CASE #diSelectCaseFilm OF
  0: // waiting state
    IF #FilmPullCam_Modification_Call THEN
      IF #iq_array_StatusWord["SM_Stretch_cFilmPull"].synchronous THEN
        IF #FilmPull_Ctrl.cam = "Film_Pull_Cam_A" THEN
          #Interpolate_FilmPull_Cam := "Film_Pull_Cam_B";
        ELSE
          #Interpolate_FilmPull_Cam := "Film_Pull_Cam_A";
        END_IF;
        #NeedChange_FilmCam := TRUE;
      ELSE
        #Interpolate_FilmPull_Cam := "Film_Pull_Cam_A";
      END_IF;
      #diSelectCaseFilm := 10;
    END_IF;

  10:
    #FilmPull_Execute := TRUE;
    IF #FilmPull_Cam_Instance.done THEN
      #FilmPull_Execute := FALSE;
      #FilmPullCam_Modification_Call := FALSE;
      #diSelectCaseFilm := 0;
      IF #NeedChange_FilmCam THEN
        "Stretch_DB".ChangeFilmPullCam := TRUE;
      ELSE
        #FilmPull_Ctrl.cam := "Film_Pull_Cam_A";
      END_IF;
      #NeedChange_FilmCam := FALSE;
    END_IF;
  END_CASE;

```

Figure 4.15: Interpolation code of the alternative cam.

## 4.9 Errors

A similar library to the Schneider error management is not available for Siemens. This implies the development of a custom routine. To do this, I developed the FB *Fault\_Signal\_Management* that allows centralizing the management of the faults of the module in which it is called. In particular, this FB is the same for both the sub-modules and, by exploiting the Siemens FBs properties, two different instances are created that make the error handling independent in the two Error blocks of the structure of the module. In addition to this, some other logic is present inside the motion control algorithm to support this management and leaving to it only the job of handling the faults. The alarm definitions, activations and successive reactions are treated as the below parts describe.

### 4.9.1 Error management FB

To start, here it is described the basics of the faults handling inside the FB in order to better understand how it is used and the exact logic that it contains. It is defined as a case block where each of the five states contained execute a precise operation.

To better understand the functionality of this management, it is important

---

to underline the most important variables that characterize the FB. Specifically, it is organized through the use of two sets of variables:

Static variables	Active variables
Signals	ActiveSignalCodes
FirstSignalIndex	FirstActiveSignalIndex
LastSignalIndex	LastActiveSignalIndex
NumberOfAlarms	NumberOfActiveAlarms
NumberOfAnomalies	NumberOfActiveAnomalies

The former is dedicated to the static storing of the information about the active faults. Indeed, the array *Signals* is an array of bits. Its dimension is set to a number larger than the maximum number of faults present in the sub-module where the function is called. This allows matching the bit in each position of the array with the code of the faults. Then, it is used to set the bit in relation to the active state of the fault itself. Consequently, the codes of the active faults are always known along with their order of importance. In addition to it, the numbers related to the first and last faults active are present. By updating them at every function call, the exact extremes of the array can be passed as output. About the latter, the active variables are dedicated to the dynamic handling of the faults. Starting from the array *ActiveSignalCodes*, as the name suggests, it contains the codes of the active faults. The main difference is that each position contains an integer and no holes are present when it is filled. In particular, this array is divided into the same number of categories present in the definition of the fault and each of them has as many positions in the array as the number of faults in the relative category. This results in a proper distinction between them. While filling the array, the new faults detected would modify only the relative category without interfering with the others. This allows more efficient management of the faults because the ordering can be considered only for one group at time. The additional variables of this set are dynamic information of this array. The first and the last indexes allow knowing exactly the extreme positions, while the numbers of active faults tell the exact number per category. In parallel to the last ones, the total number of faults per category are present. The difference resides in the fact that these tell the total number per category while the former only the active ones, These are important numbers. As will be discussed below, they are used in order to properly access the two arrays.

The block state mentioned above is an input that must be specified when the block is called. This makes the states of the FB different codes that are executed only when they are called with their number specified in the input. The states are organised in two groups: one is used inside the Initialization block, the second in the Error block. In particular:

1. the states 0 and 10 are the two used for the initialization procedure of the logic. As mentioned, the call of the FB is performed by specifying the state it must perform. The first calls are used to reset the internal

---

variables of the FB in order to clean the content. Then, the alarms defined before in the Initialization block are processed. Therefore, the first function call implies that the initial state (namely 0) is specified. Then, the next function call follows inside a "for cycle" after the state input value is changed to the second one, namely 10. Inside the "for cycle", the SignalType variable of each alarm is passed. This allows storing in the internal variables of the FB the number of alarms for each category. These numbers are going to be fundamental for the faults management later in the successive states. About this, one of the principal advantages of this approach is the possibility to define an arbitrary number of alarms. Indeed, the only requirement is to re-initialize the PLC as the block must be reset. Otherwise, no limit is set to the number of alarms that can be added.

2. the states 100, 110 and 120 contain the proper faults handling. They are the core of alarms management. Taking into consideration the state 110, it is the more relevant one. Here, the alarms are organised and the resulting reaction is set as output. Its routine can be divided into four steps:

- As a first step, the activation signal of the fault processed is checked along with the static array of signals. If the fault was activated during this cycle of the OB1, namely the activation variable is TRUE while the bit of the static array in the relative code position is not, it is added to the FB variables. Firstly the static array is updated. Then, depending on the category it belongs to, it increases the variable of active faults. By exploiting these new values, the active array is updated through a for cycle by scrolling the static one.
- The second step consists in enabling the reset. If it results that the fault was already active in the previous OB cycle, the variable that enables the reset call is activated. Namely, if a call is requested, the FB can reset the faults.
- The third step is used to update the first and the last positions of both the static and active arrays. In the first case, they are initialized both at zero. Then, scrolling the static array, they are updated. This means that the first index is modified only at the first active fault, while the last index is updated every time a new one is found. In the second case, the numbers of active faults per category is considered. For example, considering this project where only alarms and anomalies are present, firstly the *NumberOfActiveAlarms* is controlled to define the *FirstActiveSignalIndex*. If it is greater than zero, it means that a fault in the more important category is present, resulting in the first bit of the active array to be filled. Contrary, if it is zero while the *NumberOfActiveAnomalies* is not, the first element would coincide with the first position of them and so on. For the *LastActiveSignalIndex* a similar procedure

---

is applied but inverted. Starting from the number of elements in the less important category, they are processed towards the more important. These variables do not consider the difference between the faults categories. Instead, they specify the more important and the less one active.

- The last step is used for the computation of the reaction. It must be considered how the reactions are defined in order to understand this. They are defined through a binary constant variable. This one is structured to have only one bit different from zero. In this way, each reaction can be identified with a bit of a binary variable. Below in this section the faults definition is treated and also the reaction used and their meaning are present along with it. With this structure, if the fault treated by the FB is active, its reaction is added to the *OutputSignals* binary variable through an "OR operation". In this way it is possible to know at once all the reactions required by the faults.

Regarding the states 100 and 120 they are used for operations of completeness. The first is only used to reset at each new cycle of the OB1 the reset and the output variable. The second one resets the values of all the variables used in the state 110 for the fault management. This acts a total clear of the faults even in case they are not solved. This coincides with a project choice that follows from the previous machine politics. To perform such operation and to avoid useless computations, if a successful reset request has been preformed, the static and the active signals array are reset with a for cycle only if the related last index is different from zero.

### 4.9.2 Error logic

In addition to the FB, additional support logic is present. The faults are indeed handled in two distinguished parts of the code.

**Initialization block** They are at first defined inside the initialization phase in a dedicated array of structure *Generic\_Fault\_Definition*. This structure is articulated as follow:

	Variable type	Meaning
SignalType	Integer	Alarm category
SignalCode	Integer	Alarms specification code
SignalMessage	String	Synthetic alarm description
ActivationSignal	Bool	Variable that enables the error
Reaction	Dword	Procedure to be execute when the error occurs
Detail	String	More specific description of the alarm



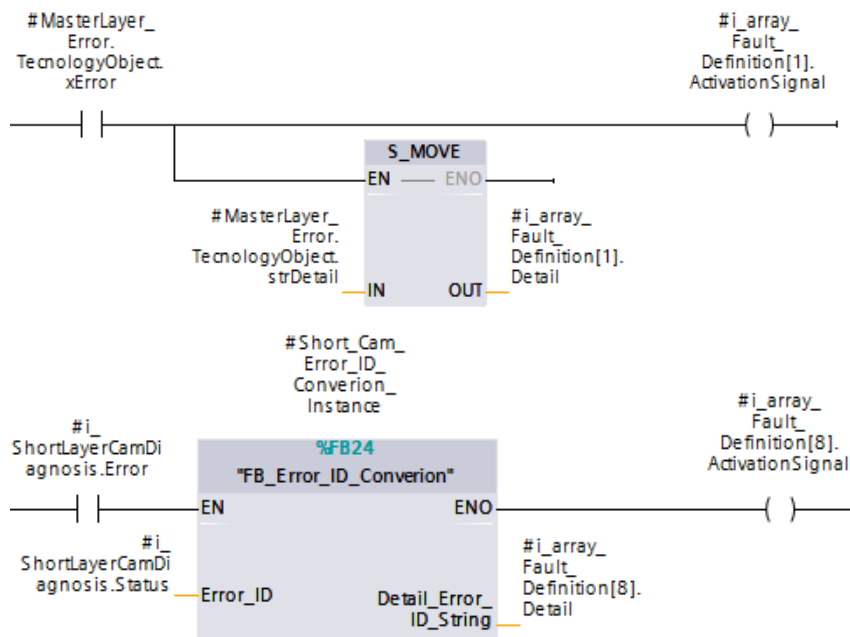
---

It allows defining an array of alarms inside each sub-module where all the information about them are saved. In this way, they can then be accessed every time it is necessary. In the description of the Initialization block before in this chapter, it is specified how the alarms are defined. There are two main categories specified in both the sub-modules: Alarms and Anomalies. The firsts are the worst events that can occur in the machine. They imply the faults of the motion parts of the control logic. This means the faults of everything linked to the axes: technology objects, drives and motion FBs of the library LAxisCtrl. The immediate stop call from the PLC is also added to them. The Anomalies category contains all the least threatening errors that could occur in the code. This means: gearbox speeds too high, fail in the computation of the cam points, fail in the cam interpolation, interference in some part of the machine, failed acquirement of axes positions from the read of the cam.

As the alarm structure shows, each fault is characterized also by a reaction. They are three: `AUTOMATIC_STOP`, `IMMEDIATE_STOP`, `PHASE_STOP`. The first is the reaction linked to the fault of the pieces of hardware linked to the axes. Indeed, when such errors occur, the Siemens software autonomously stops the motors safely. Then, the detection of these types of alarms, which all belong to the worst ones, implies an automatic reaction from them. In this sense, it is only necessary to execute the stopping procedure for the code. Namely, the code understands the presence of the automatic stop from the axes and then puts itself in a waiting state where a reset is expected to restart the operations. The `IMMEDIATE_STOP` is performed only for the last Alarm remained the immediate stop call from the PLC. In this case, the system is brought to an immediate stop of the axes by quitting the operations and re-setting the commands of the axes. The difference with respect to the previous case is how the axes are stopped. Indeed, this stop allows controlling all the motors, while the previous required to control only those that did not enter in a faulty state. The last reaction is more light. Namely, in this case its allowed for the motors to terminate their operation before be stopped. The details of how the reactions are executed are described more precisely in the motion section. Indeed, this part only describes the alarms definition, and handling and the choice of the type of reaction.

**Error FB** The real management of the faults is centralized in the second part of the code, namely the Error management block. About the definition of the alarms, it is described above how only 5 out of the 6 terms contained in the *Generic\_Fault\_Definition* structure are used. This is because the *ActivationSignal* is used exclusively inside this block. In particular, it is coded as the Axes modules by using the ladder language and the ST language. This choice was made because the first allows a better debugging and visualisation of one of the two parts that define this FB, while the second allows a simpler coding of the function call and cycles that would have been articulated to be performed in ladder.

The first part mentioned is the one that uses the *ActivationSignals*. Indeed, at the beginning of the block, all the alarms initialized are reported. More specifically, each network contains the logic that pairs the activation variables with the states of the algorithm that should enable them. Depending on the type, each network activates the error in a different way. Fig. below shows the three different cases. For the elements linked to the motors, the code only looks at the relative error variable generated by the Axis module (here the Master of the layer motor is considered). This enables the *ActivationSignal* of the alarm linked by reference to its position in the array of structure and in parallel it enables also the Word-To-String conversion of the detail number that the FB gives. This last operation is necessary for the information that are reported along with the alarm that is going to be shown later. The second case is related to the excessive speed of the Layer gearbox. This is computed with the specific *FB\_GearboxSpeedCheck* described in one of the previous sections. The last case is analog to the first. This is dedicated to the fault of the interpolation for one of the Layer axis cams.



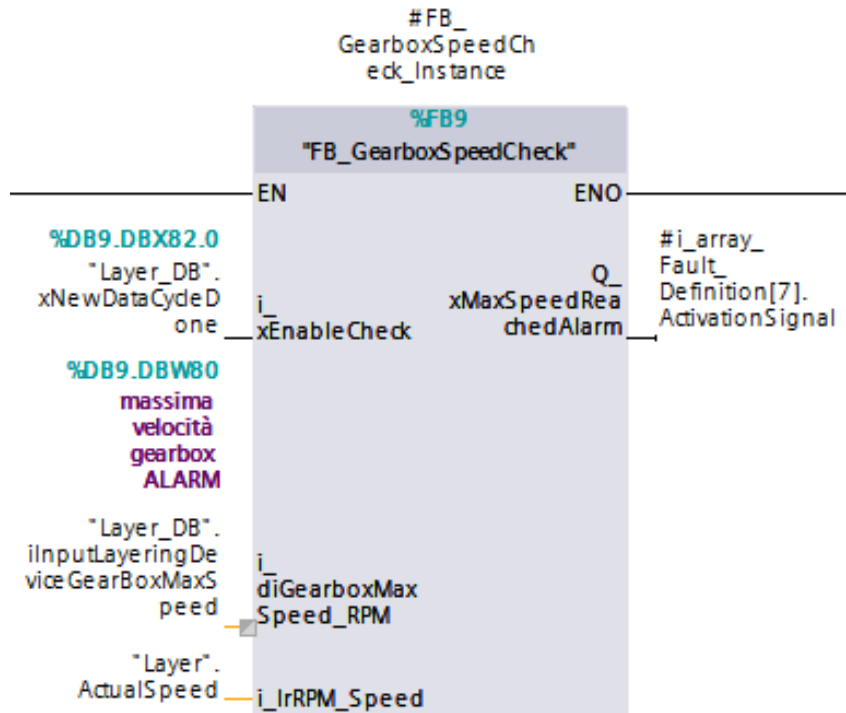


Figure 4.16: Fault activation examples.

The second part of the Error block, analogously to the initialization block, is where the fault management custom FB *Fault\_Signal\_Management* is called. As the first instruction, the reset request is performed. It remains active only for one cycle. This implies the use of a supporting variable that is updated every time the physical button is pressed. Before this, the FB input is modified through the "OR operation" between the button variable itself and the negated value of the support one. In this way, the request is passed only in case the physical input was FALSE in the previous cycle. After this, the FB call is performed. Firstly, specifying the state 100 for the preparation of the new cycle. Then, with the state 120 for the reset in case the request was successfully performed. Lastly, state 110 is specified in the input of the FB. In this way, the call of the FB that follows inside a "for cycle" would bring the new inputs to that state. In this part, indeed, all the faults are processed by the FB in order to establish the activation of the new ones and the reaction requests of the active ones. In addition to this, the creation of the environment for the execution of the reactions takes place. After the FB calls, the output variable related to the reactions is filtered in order to update the reaction variables. This operation is performed by checking if the reaction constants, used for the definition of the faults in the Initialization block, are equal to the filtering through an "AND operation" between the output and the constant itself. For example, the phase stop request is performed as follows:

$$((OutputSignals \text{ AND } "ON\_PHASE\_STOP") = "ON\_PHASE\_STOP");$$

---

In case one of the stop requests is active, a variable is updated. This allows the operative motion blocks described above to quit their operations and enter in the stop phase depending on the reaction active. Moreover, the properties of the more important faults active must be notified. In particular, the code, the reaction, the message and the detail are passed to the output variables of the Error block. These are then used in two ways. One is to send the data to the HMI panel for notification to the operator. The second is to use those information for the other modules. Indeed, the reaction and the presence of a fault in one sub-module are shared with the other one. This allows generating a reaction even if a fault did not occur. In particular, this always implies the activation of the phase stop.

## 4.10 HMI panel

The HMI is the bridge between the operator and the machine. It allows to send commands to the PLC and to receive the status feed-back and the operative conditions of the axes and sensors. The panel is a TP900 Comfort Panel. Therefore, the code of the HMI is still implemented in the TIA Portal software. As the project is developed in a different file with respect to the PLC and motion control one, it is necessary to insert an instance of the PLC itself. Then, in the project views (topology and network) a communication link between the two must be created. In this way, it is possible to configure the IP address of the panel in order to allow a connection path. The HMI code is not so much influenced as it is the PLC from the new motion control algorithm. In this case, indeed, only a few parameters had to be changed, specifically related to the new functionalities and management of the code. The more relevant parts that required a new code development are the two brand new functionalities added to the MS260. They are the cam profile viewer and the Timer-System DB import and export. Both of them required the creation of new pages in the HMI in order to exploit the new codes generated.

**Cam show** The Cam show functionality allows printing in a dedicated page for each axis the cam currently used by them. It performs also the computation of the velocity and the acceleration profiles linked to the cam profile, making them available on the HMI. The execution of this functionality is characterized exclusively by a FB used inside the format change action of the sub-modules. This relies on the utilization of a single FB call in the actions that is inserted outside of the case. This allows performing the block call even if a format change is not present, leaving this functionality always available. This FB allows, in the first place, to perform the change of the cam selected and the computation of the cam points to be shown. Their trigger relies on the activation of the buttons on the dedicated HMI page. In Fig. 4.17 on the next page it shows the structure of the page dedicated to all the real axes of the motion control, but in this case the one dedicated to the Layer axis is reported. The two buttons on the top allows enabling the two operations above mentioned. The square below shows the DB number of the cam currently selected and the

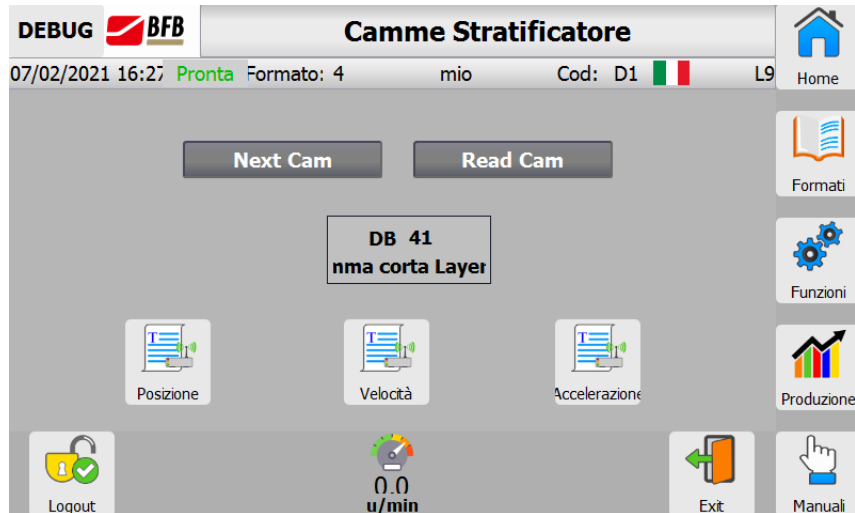


Figure 4.17: Show cam page.

name was given to it. Then, the remaining buttons bring to different pages where the profile of the cam is depicted. Three alternatives are possible: the position, the velocity and the acceleration trajectories.

The first operation is performed when the button *Next Cam* is pressed. It activates the input *cam1or2switch* of the FB. From this, the internal FB *HMICamHdl.ChangeCam* allows performing the change of the cam selected giving to the DB number and the name of the new ones. This FB called in the *SM\_Layering* is anticipated by an additional piece of code. The cam-change allows to perform only the switch between two cams that are passed to the input. However, in the case of the *Layer*, it is characterized by three possible cams. Therefore, the HMI button does not trigger the FB input directly. Instead, it increases a counter with a maximum value of 3 which allows passing to the FB inputs the related cam information, namely: DB number, name, cam DB and cam TO.

The second operation is more complex. When the *Execute* is triggered, the FB computes the following information related to the cams: the values of master and slave positions, values of slave velocity and acceleration, minimum and maximum of position velocity and acceleration. In this way, the HMI graphs can show the slope of each trajectory with the values of minimum and maximum at the extremes.

**DB import and export** This functionality allows moving the parameters of the system and of the timers DBs between the SD card of the PLC and the PLC memory. Thanks to this, it is possible to copy these two sets of variables in a dedicated text file on the SD card. Moreover, the inverse movement can be performed. The data defined in a target text file can be read and stored in the DBs variables as long as the structure of the file is correct. These operations are divided between some instructions performed in the PLC and others in the HMI.

The former is dedicated to the management of the DBs variables. They,

---

indeed, must be stored in a support array that is then used for the variable transmission with the HMI. These operations are performed by a single FB called *Hmi\_FileConnection*. It is called at the end of the OB1 as it does not depend on the functionality of any part of the machine and can be performed at every time. Fig. 4.18 on the following page shows the FB call with the relative inputs and outputs. In particular, apart from the Execute that is used for the block activation, the others are configuration variables. *Select\_Sys\_DB* and *Select\_Timer\_DB* are used to select on which DB the operations must be performed and are exclusively inputs. The *WriteDataToHMI* is used to select the operation with which the DBs are written on the text file, otherwise, they are read from it. The *Array\_Dimension* and *Array* are input-output variables and are used for the communication of the data with the HMI. Through them, the PLC can communicate with the HMI the array of data and its dimension and the other way around. The two outputs *Done* and *Error* are only used for debugging purposes. Inside the FB, a case logic is used. The initial state is the organization one. Here, the activation of the Execute is detected. As it is not an Enable, the operations are terminated, namely the outputs are reset and a new cycle can be performed, when a negative value of it is detected. The cycle, instead, starts only with a positive edge. Then, in the same state, the operation is driven to the one to be performed on the base of the configuration inputs. In any case, the DB dimension is computed in order to initialize the "for cycle" index. Within the two DBs there can be many parameters and these values must be copied into an array; this is particularly necessary for the Timer DB because the type "timer" used for the definition of its elements must be converted to an integer in order to be used for the export. Therefore, keeping stuck the OB1 in a cycle for many computations is not ideal. This is not an essential operation to be performed and therefore the cycle is divided into smaller ones of a maximum number of 19.

The cycle states are one for each circumstance: write and read of the system parameters and write and read for the timers. The writing and reading operations are slightly different between them. For the former, firstly the "for cycle" is performed for all of the 19 elements or until the end value is reached. In the end, if the index is equal to the array dimension, the dimension is passed to the input-output variable and the case passes to the end state. Otherwise, the cycle extremes are updated to the successive 19 parameters. For the reading case, an initial state is used to detect if discrepancies are present between the HMI and the PLC dimensions. In case, an error is raised otherwise a similar state to the previous case is executed; the "for" is performed until the end and the extremes are updated to the next 19 parameters before switching to the ending state. There, the done is raised and the case returns to the initial state.

The HMI operations are performed through scripts that are executed when the button of the page is pressed. Fig. 4.19 on page 143 shows how it is organised. On the left, the buttons trigger the export of the DBs to the text files, while on the right the inverse operation. The circle in the center allows understanding when the computations are terminated.

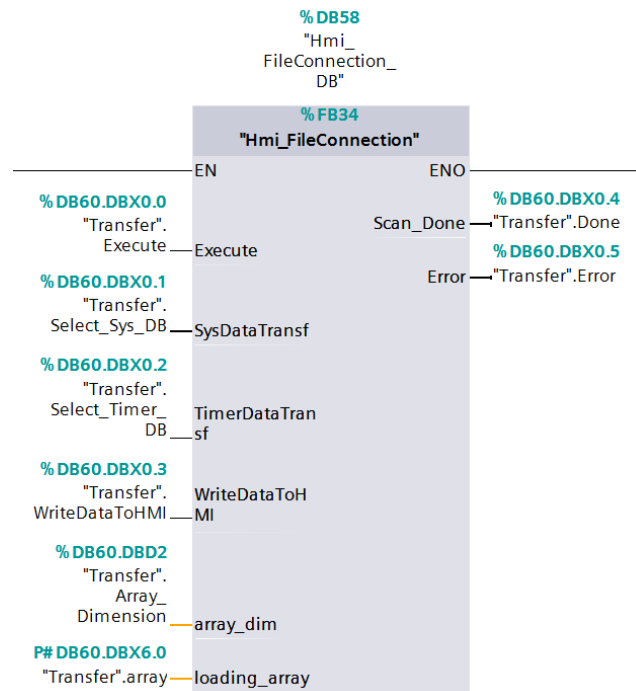


Figure 4.18: PLC FB for DB import-export.

The former is performed through a single VB script. The inputs are defined while the button is pressed. Indeed, the VB script allows to set them depending on where it is called. Therefore, the difference between the system DB readings and the timer DB reads depends on the definition of the input. At the beginning of the script, the PLC FB configuration inputs are defined on the base of the script inputs. Then, a waiting command is started in order to let the PLC terminate the loading of the array. The wait ends when the Done/Error is raised by the FB or the time expired. In a successful scenario, the script continues. Then, a "Xor" debugging is performed. This operation consists of the creation of an array composed of elements where everyone comes from the "Xor" operation between one DB parameter of the just loaded array and a filter variable. This is required essentially for the inverse operation in order to avoid the direct copy of the values of the text file into the sensitive software DBs. If differences are detected between the DB values and the file values, consent to continue the operation is required. This is particularly useful if the structure of the file is not the one expected. Lastly, the text file is created, if not present, and it is loaded with the array elements.

The reading operation is articulated between more scripts. The first one is the one called from the press of the buttons and contains the configuration values definition. In the beginning, the script checks the existence of the file to be read. Then, the lines of the file are stored and decomposed and the values are copied in a temporary array. This allows performing the "Xor" check previously described. If an operation was not performed on the DB under analysis, namely the "Xor array" is empty, the writing script is called to be executed and it ends immediately after the "Xor check" without the writing

of the file. In this way, the text parameters can be controlled. As soon as one difference is detected, the control is aborted and a pop-up page is shown on the panel. It asks for the continuation of the operation. If a positive answer is given, a new script is called. Here, the "Xor check" is restarted in order to store the new filtered values in the "Xor array". In the end, the last script is executed. Finally, it performs the copy of the file parameters to the transfer array that will be used by the PLC FB. This operation is more complex than the previous one. Indeed, the bus can run into an overflow if too many values are passed by the "for cycle". Therefore, only 50 cycles are executed before introducing a wait of 300 ms. Then, the Execute of the PLC FB is raised and after its termination, it is reset to be ready for the next operation call.

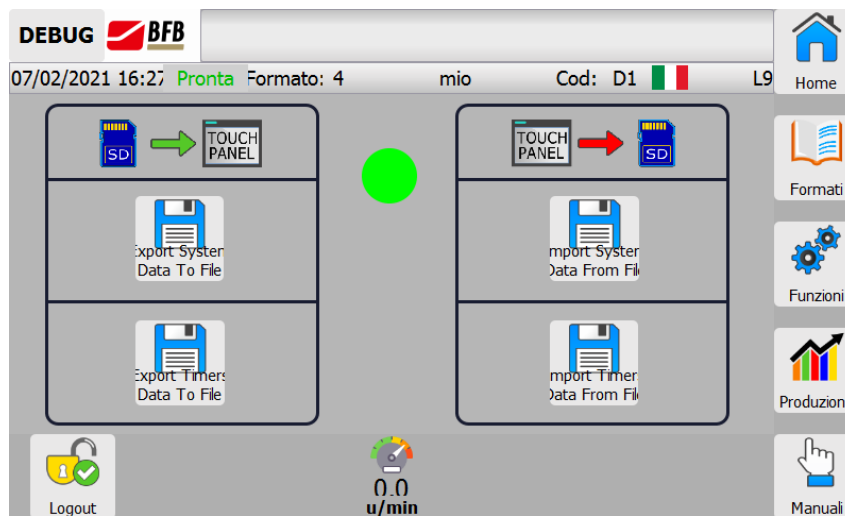


Figure 4.19: HMI DB import-export page.

## 4.11 Simulation

At the end of the development, some tests are performed. Unfortunately, since the drives and the actuators were not available, a real machine on which performing them was not present. Therefore, the tests are actually simulations that allow checking the correct functionality of the algorithm. To do them, two are the tools available: the simulation of TIA Portal (called SIM-Advanced) and the PLC. Indeed, the Simatic et200sp Open Controller selected as new PLC was kindly provided by Siemens for this purpose. Therefore, the tests that can be done are:

- Hardware in the Loop:  
HIL is a technique for validating the control algorithm, running on an intended target controller, by creating a virtual real-time environment that represents your physical system to control. It helps to test the behavior of the control algorithms without physical prototypes.
- Software in the Loop:  
A SIL simulation compiles the generated source code and executes the



---

code as a separate process on the host computer.

The former is possible thanks to the PLC provided, while the latter is always possible with the TIA Portal simulator. They have significant differences and are necessary both to understand different aspects. Two different projects are generated. One is defined by the Simatic et200sp Open Controller but no additional hardware is added. The second is characterized by a different PLC (S7-1515T) and the actuators are also inserted. This substantial difference comes from the related two necessities.

For the HIL simulation, the PLC is directly connected to the PC through the Ethernet ports. So, the additional hardware of the drive and the motors that should be contained in the project, are in reality not connected to the physical ports of the PLC. Therefore, the PLC itself would generate an error. From this comes the necessity of the first project where, consequently, the TO axes are all defined as virtual.

For the SIL simulation, instead, the second project is mandatory. The PLC used is an equivalent of the Open Controller with the substantial difference that it is more simple. As it is described in the second chapter, the Simatic et200sp Open Controller is characterized by a side component where Windows OS is installed, making it possible to connect a screen directly to it. For this, the simulation of such PLC through SIM-Advanced is not possible. So, the S7-1515T is mandatory for the SIL simulations. It has similar abilities to handle several TOs while having comparable speeds. In the project, then, the hardware can be added and the axes connected to their encoders considering as real and in simulation and not virtual.

Also, the HMI is simulated. It helps to visualize the machine status and to give the commands for the operations. So, for both the tests, the TIA Portal allows to run it with the internal simulation tool of the software in parallel to the PLC. The HMI project comes also in two forms. In it, indeed, an instance of the PLC must be added and in the project views, it must be connected to the panel. Therefore, to use the HMI in both the simulations two of them must be present. To complete the simulation, bypass variables are inserted in the PLC code regarding the physical inputs and outputs of the sensors connected. In this way, the operative conditions of the machine can be forced to act as they should.

From the simulations, only a few but important results can be get. Indeed, the correct functionality of the motion algorithm coincides with the expected behavior. In particular, the nominal automatic mode, the bypass mode and the manual operations were all successfully executed. However, even with the possibilities described above, something about the algorithm could not be tested. They are the cleaning cycle of the sealer and the drive errors. The former requires the response from the safety system implement in the machine. It is characterized by physical contacts that react to certain events. It is impossible to force the expected behavior of them while the sealer cleaning steps are followed. About the latter, not having the direct response from the drives, possible faults that can come from them and their reset can not be

---

simulated. Examples of successful simulations are the Figures of cam profiles presented in chapter four. They are the result of the format change completed without errors. During the tests, it was also possible to look at the correct behavior of the alarms system. On a dedicated page of the HMI, the properties of the active faults are visible and the subsequent reset allows a restart of the operations.

Regarding the PLC, Fig. 4.20 shows its performances. In particular, the most important is the duration of the cycle. The average length is less than one millisecond during operating conditions making it really fast despite the number of axes to be controlled. About this, the values on the bottom show how loaded is the memory.

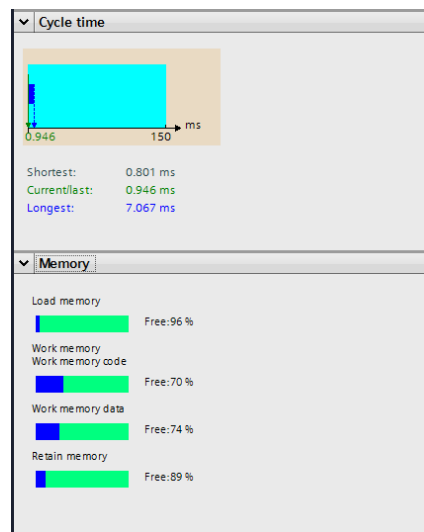


Figure 4.20: PLC performances.

# Conclusions

## Main results

This project has the main objective of creating a machine completely built and controlled by Siemens. A machine of this type can offer a lot of advantages.

Firstly, the Siemens catalog presented in the second chapter gives a larger choice about the actuators to be used. Very different motors can be mounted on a machine generating a perfect sizing for the application needed. This aspect, however, suffers from an economic and technical drawback. The cost of the motors is variant concerning the technology used and the dimension with which it comes but is not true for all the cases. So, the use of smaller motors for applications that requires lower specifics could allow a reduction of the costs but not so significantly with respect to a larger one. In addition to this, the application of different types of motors encounters the necessity of a greater part of the industrial warehouse dedicated to the replacement parts. So, the possibility to have specifically designed motors must be analysed also under other technical aspects.

Then, the comparisons between the old algorithm presented in the first chapter and the new one in the fourth chapter allow finding the real benefits of a project of this type. The simpler and more intuitive aspect is that the software complexity is greatly reduced. The debugging and the code management became more direct. Indeed, being the code fully contained in a single project simplifies, in first place, its handling. Moreover, from this aspect comes out also a more technical reason. By running the machine control algorithm in one computer center instead of two, the probabilities of fault are restricted to the single PLC.

Regarding, instead, the technical data, the unification of the motion control and the machine status management gives a faster communication channel between the two parts. Having the data always stored in the PLC memory makes faster and safer the management of the motion control data and, therefore, the execution of the operations by the motors. The eventual drawback of a too much loaded PLC for the execution of both the projects is not present. The HIM simulation discussed in the last chapter shows the really good performances in terms of milliseconds per cycle of the PLC under these circumstances.

---

## Future developments

On one hand, the next phase of this project relies on the tests on the real machine. Once it is built, the last functionalities must be analysed to check their correct execution. Then, the machine can be considered complete. Regarding the PLC used, the Windows OS leaves open the possibility of a new type of HMI. The software can, indeed, be generated and executed inside Windows and, exploiting the OS functionalities as a distinct PC, a dedicated panel for the execution of the interface can become an excess.

On the other, at IMA BFB, many types and versions of machines that are controlled likely the MS260 considered are present. Therefore, this same concept can be applied to the other machines to benefit from the same aspects. Indeed, the structure used for this goal was built keeping a modular structure. This allows exporting and adapting the motion control to other types of operations, making this project the reference point for the new ones that could be converted to the motion Siemens.

# Appendix A

## Crank module code

In this appendix, the code realized for the Crank module library is reported. As described in chapter three, this library contains three FCs and one FB. Below the code is divided for each function described in the section related to keeping the same order used.

### Rotation to Translation FC

```
#PI := ACOS_LREAL(-1);
#tempVerticalOffsetAbsolute := ABS_LREAL(#Crank_Data.Vertical_Offset); // Value is often needed

// Check input parameters
IF (#Crank_Data.Radius_Lenght <= 0.0) OR (#Crank_Data.Rod_Lenght <= 0.0) THEN
  #RotationToTranslation_Compact := #ERR_CRANK_OR_CONROD_LENGTH_INVALID;
  RETURN;
ELSEIF #tempVerticalOffsetAbsolute > (#Crank_Data.Radius_Lenght + #Crank_Data.Rod_Lenght) THEN
  #RotationToTranslation_Compact := #ERR_VERTICAL_OFFSET_INVALID;
  RETURN;
END_IF;

REGION CALCULATE_MIN_MAX_ANGLE
// 360.0° rotation is not possible
IF ABS_LREAL(#Crank_Data.Rod_Lenght - #Crank_Data.Radius_Lenght) < #tempVerticalOffsetAbsolute
THEN
  IF (#Crank_Data.Vertical_Offset >= 0.0) THEN // Jaws are under abszisse
    #tempAngleMin := (ASIN_LREAL((#Crank_Data.Vertical_Offset - #Crank_Data.Rod_Lenght)
      / #Crank_Data.Radius_Lenght) * 180.0 / #PI);
    #tempAngleMax := (180.0 - #tempAngleMin);
  ELSE // Jaws are at top of abszisse
    #tempAngleMin := (180.0 + ASIN_LREAL((#tempVerticalOffsetAbsolute -
      #Crank_Data.Rod_Lenght) / #Crank_Data.Radius_Lenght) * 180.0 / #PI);
    #tempAngleMax := (360.0 - ASIN_LREAL((#tempVerticalOffsetAbsolute -
      #Crank_Data.Rod_Lenght) / #Crank_Data.Radius_Lenght) * 180.0 / #PI);
  END_IF;
// 360.0° rotation is possible, each angle is valid
ELSIF (ABS_LREAL(#Crank_Data.Rod_Lenght - #Crank_Data.Radius_Lenght) >= #tempVerticalOffsetAbsolute)
AND (#Crank_Data.Rod_Lenght > #Crank_Data.Radius_Lenght) THEN
  #tempAngleMin := (0 - ASIN_LREAL(#Crank_Data.Vertical_Offset /
    (#Crank_Data.Rod_Lenght - #Crank_Data.Radius_Lenght)) * 180.0 / #PI); //0.0;
  #tempAngleMax := (180 - ASIN_LREAL(#Crank_Data.Vertical_Offset /
    (#Crank_Data.Rod_Lenght + #Crank_Data.Radius_Lenght)) * 180.0 / #PI); //360.0;
// 360.0° rotation is not possible
ELSIF (ABS_LREAL(#Crank_Data.Rod_Lenght - #Crank_Data.Radius_Lenght) >= #tempVerticalOffsetAbsolute)
AND (#Crank_Data.Rod_Lenght < #Crank_Data.Radius_Lenght) THEN
  #tempAngleMin := (180.0 - ASIN_LREAL((#Crank_Data.Vertical_Offset + #Crank_Data.Rod_Lenght) /
    #Crank_Data.Radius_Lenght) * 180.0 / #PI);
  #tempAngleMax := (180.0 + ASIN_LREAL((#Crank_Data.Rod_Lenght - #Crank_Data.Vertical_Offset) /
    #Crank_Data.Radius_Lenght) * 180.0 / #PI);
```

```

// Write corresponding output parameters
#angleMin := #tempAngleMin;
#angleMax := #tempAngleMax;
END_REGION

REGION CALCULATE_POSITION
// Angle to be converted exceeds calculated range
IF (#angle < #tempAngleMin) OR (#angle > #tempAngleMax) THEN
    #RotationToTranslation_Compact := #ERR_ANGLE_EXCEEDS_CALCULATED_MIN_MAX;
    RETURN;
END_IF;

#tempSINofCrank := (#Crank_Data.Radius_Lenght * SIN_LREAL(#angle * #PI / 180.0));
#tempCOSofCrank := (#Crank_Data.Radius_Lenght * COS_LREAL(#angle * #PI / 180.0));
#tempValue0 := (#Crank_Data.Rod_Lenght * #Crank_Data.Rod_Lenght -
    (#Crank_Data.Vertical_Offset - #tempSINofCrank) *
    (#Crank_Data.Vertical_Offset - #tempSINofCrank));

IF (#tempValue0 <= 0.0) THEN
    #RotationToTranslation_Compact := #ERR_SET_OF_INPUT_PARAMETERS_INVALID;
    RETURN;
ELSE
    // Write corresponding output parameter
    #tempRealPos := SQRT_LREAL(#tempValue0) - #tempCOSofCrank;
    #position := #tempRealPos - #Crank_Data.Start_Offset;
END_IF;
END_REGION

```

## Translation to Rotation FC

```

#crank_linear_position := #Real_Linear_Pos + #Crank_Data.Start_Offset;
#tempVerticalOffsetAbsolute := ABS_LREAL(#Crank_Data.Vertical_Offset); // Value is often needed

// Check input parameters
IF (#Crank_Data.Radius_Lenght <= 0.0) OR (#Crank_Data.Rod_Lenght <= 0.0) THEN
    #TranslationToRotation_Compact := #ERR_CRANK_OR_CONROD_LENGTH_INVALID;
    RETURN;

ELSIF #tempVerticalOffsetAbsolute > (#Crank_Data.Radius_Lenght + #Crank_Data.Rod_Lenght) THEN
    #TranslationToRotation_Compact := #ERR_VERTICAL_OFFSET_INVALID;
    RETURN;
END_IF;

REGION CALCULATE_MIN_MAX_POS
IF (#Crank_Data.Rod_Lenght >= #Crank_Data.Radius_Lenght) AND
    (#Crank_Data.Rod_Lenght - #Crank_Data.Radius_Lenght > #tempVerticalOffsetAbsolute)
THEN // 360° rotation possible
    #tempLinearMax := SQRT_LREAL((#Crank_Data.Rod_Lenght + #Crank_Data.Radius_Lenght) *
        (#Crank_Data.Rod_Lenght + #Crank_Data.Radius_Lenght) -
        #Crank_Data.Vertical_Offset * #Crank_Data.Vertical_Offset);
    #tempLinearMin := SQRT_LREAL((#Crank_Data.Rod_Lenght - #Crank_Data.Radius_Lenght) *
        (#Crank_Data.Rod_Lenght - #Crank_Data.Radius_Lenght) -
        #Crank_Data.Vertical_Offset * #Crank_Data.Vertical_Offset);

ELSIF ((#Crank_Data.Rod_Lenght >= #Crank_Data.Radius_Lenght AND
    (#Crank_Data.Rod_Lenght - #Crank_Data.Radius_Lenght <= #tempVerticalOffsetAbsolute)) OR
    ((#Crank_Data.Rod_Lenght <= #Crank_Data.Radius_Lenght AND
    (#Crank_Data.Radius_Lenght - #Crank_Data.Rod_Lenght <= #tempVerticalOffsetAbsolute)))
THEN // 360° rotation not possible
    #tempLinearMax := SQRT_LREAL((#Crank_Data.Rod_Lenght + #Crank_Data.Radius_Lenght) *
        (#Crank_Data.Rod_Lenght + #Crank_Data.Radius_Lenght) -
        #Crank_Data.Vertical_Offset * #Crank_Data.Vertical_Offset);
    #tempLinearMin := (- #tempLinearMax);

ELSIF (#Crank_Data.Rod_Lenght < #Crank_Data.Radius_Lenght) AND
    (#Crank_Data.Radius_Lenght - #Crank_Data.Rod_Lenght > #tempVerticalOffsetAbsolute)
THEN // 360° rotation not possible
    #tempLinearMax := SQRT_LREAL((#Crank_Data.Rod_Lenght + #Crank_Data.Radius_Lenght) *
        (#Crank_Data.Rod_Lenght + #Crank_Data.Radius_Lenght) -
        #Crank_Data.Vertical_Offset * #Crank_Data.Vertical_Offset);
    #tempLinearMin := SQRT_LREAL((#Crank_Data.Radius_Lenght * #Crank_Data.Radius_Lenght) -
        (#Crank_Data.Rod_Lenght + #Crank_Data.Vertical_Offset) *
        (#Crank_Data.Rod_Lenght + #Crank_Data.Vertical_Offset));

ELSE // Invalid set of input parameters
    #TranslationToRotation_Compact := #ERR_SET_OF_INPUT_PARAMETERS_INVALID;
    RETURN;
END_IF;

```

```

// Write corresponding output parameters
#linearMin := #tempLinearMin;
#linearMax := #tempLinearMax;
END_REGION
REGION CALCULATE_ANGLES
// Position to be converted exceeds calculated traversing range
IF (#crank_linear_position < #tempLinearMin) OR (#crank_linear_position > #tempLinearMax) THEN
    #TranslationToRotation_Compact := #ERR_POSITION_EXCEEDS_CALCULATED_MIN_MAX;
    RETURN;
END_IF;

#tempHypotenuse := SQRT_LREAL(#Crank_Data.Vertical_Offset * #Crank_Data.Vertical_Offset +
    #crank_linear_position * #crank_linear_position);

IF (#tempHypotenuse <> 0.0) THEN
    // Angle between hypotenuse and abszisse
    #tempAngleHypAbsz := ASIN_LREAL(#Crank_Data.Vertical_Offset / #tempHypotenuse) * 180.0 / #PI;

    // Angle between hypotenuse and crank
    #tempAngleHypCrank := ACOS_LREAL(((#Crank_Data.Radius_Lenght * #Crank_Data.Radius_Lenght +
        #tempHypotenuse * #tempHypotenuse) - #Crank_Data.Rod_Lenght *
        #Crank_Data.Rod_Lenght) /
        (2.0 * #Crank_Data.Radius_Lenght * #tempHypotenuse)) * 180.0 / #PI;

    IF (#crank_linear_position >= 0.0) THEN
        #tempAngle1 := (180.0 - #tempAngleHypCrank - #tempAngleHypAbsz);
        #tempAngle2 := (180.0 + #tempAngleHypCrank - #tempAngleHypAbsz);

    ELSIF (#crank_linear_position < 0.0) AND (#Crank_Data.Vertical_Offset >= 0.0) AND
        (#tempAngleHypAbsz > #tempAngleHypCrank) THEN
        #tempAngle1 := (#tempAngleHypAbsz + #tempAngleHypCrank);
        #tempAngle2 := (#tempAngleHypCrank - #tempAngleHypAbsz);

    ELSIF (#crank_linear_position < 0.0) AND (#Crank_Data.Vertical_Offset >= 0.0) AND
        (#tempAngleHypAbsz < #tempAngleHypCrank) THEN
        #tempAngle1 := (#tempAngleHypAbsz + #tempAngleHypCrank);
        #tempAngle2 := 360.0 - (#tempAngleHypCrank - #tempAngleHypAbsz);

    ELSIF (#crank_linear_position < 0.0) AND (#Crank_Data.Vertical_Offset < 0.0) AND
        (ABS_LREAL(#tempAngleHypAbsz) > #tempAngleHypCrank) THEN
        #tempAngle1 := (360.0 + #tempAngleHypCrank + #tempAngleHypAbsz);
        #tempAngle2 := (360.0 + #tempAngleHypAbsz - #tempAngleHypCrank);

    ELSIF (#crank_linear_position < 0.0) AND (#Crank_Data.Vertical_Offset < 0.0) AND
        (ABS_LREAL(#tempAngleHypAbsz) < #tempAngleHypCrank) THEN
        #tempAngle1 := (360.0 + #tempAngleHypCrank + #tempAngleHypAbsz);
        #tempAngle2 := #tempAngleHypAbsz + #tempAngleHypCrank;

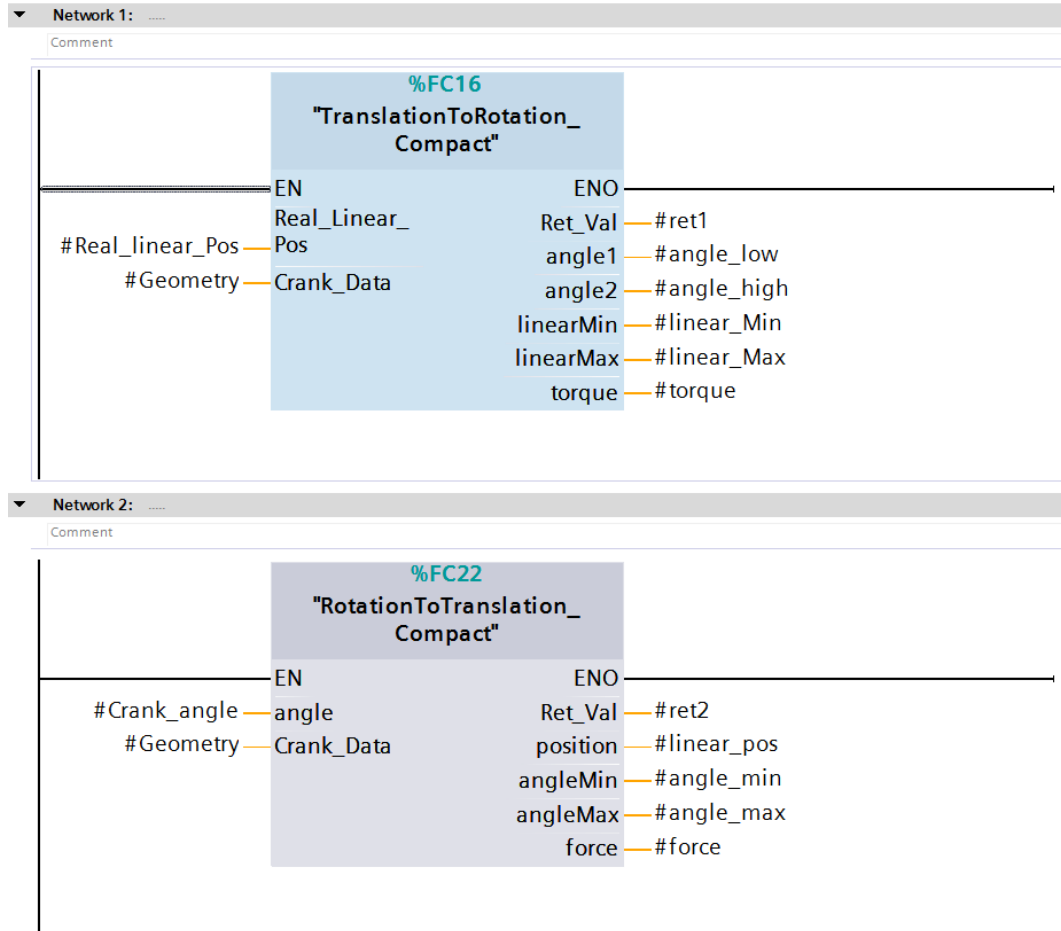
    ELSE // Invalid set of input parameters
        #TranslationToRotation_Compact := #ERR_SET_OF_INPUT_PARAMETERS_INVALID;
        RETURN;
    END_IF;

ELSE // Invalid set of input parameters
    #TranslationToRotation_Compact := #ERR_SET_OF_INPUT_PARAMETERS_INVALID;
    RETURN;
END_IF;

// Write corresponding output parameters
#angle1 := #tempAngle1;
#angle2 := #tempAngle2;
END_REGION

```

## Crank conversion FC





## Polynomial FB for cam point generation

```

//Calculation OF speed AND accelerations on slopes as a function OF the max master speed
#SlaveStartVelSlope := #Segment.SlaveStartVel / #Segment.MasterVelocity;
#SlaveStartAccSlope := #Segment.SlaveStartAcc / (#Segment.MasterVelocity * #Segment.MasterVelocity);
#SlaveEndVelSlope := #Segment.SlaveEndVel / #Segment.MasterVelocity;
#SlaveEndAccSlope := #Segment.SlaveEndAcc / (#Segment.MasterVelocity * #Segment.MasterVelocity);

//Poly5 parameter calculation
#T := #Segment.MasterEndPos - #Segment.MasterStartPos;
#a0 := #Segment.SlaveStartPos;
#a1 := #SlaveStartVelSlope;
#a2 := 0.5 * #SlaveStartAccSlope;
#a3 := (1 / (2 * #T * #T * #T)) * (20 * (#Segment.SlaveEndPos - #Segment.SlaveStartPos) -
(8 * #SlaveEndVelSlope + 12 * #SlaveStartVelSlope) * #T - (3 * #SlaveStartAccSlope -
#SlaveEndAccSlope) * #T * #T);
#a4 := (1 / (2 * #T * #T * #T * #T)) * (30 * (#Segment.SlaveStartPos - #Segment.SlaveEndPos) +
(14 * #SlaveEndVelSlope + 16 * #SlaveStartVelSlope) * #T + (3 * #SlaveStartAccSlope -
2 * #SlaveEndAccSlope) * #T * #T);
#a5 := (1 / (2 * #T * #T * #T * #T * #T)) * (12 * (#Segment.SlaveEndPos - #Segment.SlaveStartPos) -
6 * (#SlaveEndVelSlope + #SlaveStartVelSlope) * #T + (#SlaveEndAccSlope - #SlaveStartAccSlope) *
#T * #T);

#Tinterval := 1;
#tLastIndex := #T;
#FirstBasePoint := #Segment.MasterStartPos;

//Calculation OF Cam points
#NotStoredPoint_Cnt := 0;
IF #FirstBasePoint = 0 THEN
    #RotativeIndex := 0;
    #Cam_Track.LastIndex_Rotative := 0;
ELSE
    #RotativeIndex := #Cam_Track.LastIndex_Rotative;
END_IF;

FOR #tIndex := 0 TO #tLastIndex DO
    #tx := #tIndex * #Tinterval;
    #Cam_Track.Points.Linear[#FirstBasePoint + #tIndex].x := #Segment.MasterStartPos + #tx;
    #Cam_Track.Points.Linear[#FirstBasePoint + #tIndex].y := #a0 + (#a1 * #tx) + (#a2 * #tx * #tx) +
    (#a3 * #tx * #tx * #tx) + (#a4 * #tx * #tx * #tx * #tx) +
    (#a5 * #tx * #tx * #tx * #tx * #tx);

    #Command_Trasl.CALCULATE_ANGLES := TRUE;
    #Command_Trasl.CALCULATE_MIN_MAX_ANGLE := TRUE;
    "Crank_Translations"(Real_linear_Pos:=#Cam_Track.Points.Linear[#FirstBasePoint + #tIndex].y,
    Crank_angle:=0,
    ret1=>#Tr_To_Rot_Error,
    angle_low=>#Crank_Angle_Lower,
    angle_high=>#Crank_Angle_Higher,
    linear_Min=>#linearMin,
    linear_Max=>#linearMax,
    torque=>#Torque,
    ret2=>#Rot_To_Tr_Error,
    linear_pos=>#linearpos,
    angle_min=>#Angle_Min,
    angle_max=>#Angle_Max,
    force=>#Force,
    Geometry:=#Geometry);

    IF ((#Crank_Angle_Lower < #Angle_Min + #Window) OR ((#Crank_Angle_Lower > (#Angle_Max - #Window)) AND
    (#Crank_Angle_Higher < (#Angle_Max + #Window))) OR (#Crank_Angle_Higher > (360 + #Angle_Min - #Window)))
    AND ((#tIndex <> 0) AND (#tIndex <> 1) AND (#tIndex <> #tLastIndex - 1) AND (#tIndex <> #tLastIndex))
    THEN
        #NotStoredPoint_Cnt := #NotStoredPoint_Cnt + 1;
    ELSE
        #Cam_Track.Points.Rotative[#RotativeIndex + #tIndex - #NotStoredPoint_Cnt].x := #Segment.MasterStartPos +
        #tx;
        IF #Crank_360 AND #tIndex <> 0 THEN
            IF #Cam_Track.Points.Linear[#FirstBasePoint + #tIndex].y <
            #Cam_Track.Points.Linear[#FirstBasePoint + #tIndex - 1].y THEN
                #Cam_Track.Points.Rotative[#RotativeIndex + #tIndex - #NotStoredPoint_Cnt].y := #Crank_Angle_Higher;
            ELSE
                #Cam_Track.Points.Rotative[#RotativeIndex + #tIndex - #NotStoredPoint_Cnt].y := #Crank_Angle_Lower;
            END_IF;
        ELSE
            #Cam_Track.Points.Rotative[#RotativeIndex + #tIndex - #NotStoredPoint_Cnt].y := #Crank_Angle_Lower;
        END_IF;
    END_IF;
END_FOR;

#Cam_Track.LastIndex_Linear := #FirstBasePoint + #tIndex;
#Cam_Track.LastIndex_Rotative := #RotativeIndex + #tIndex - #NotStoredPoint_Cnt;

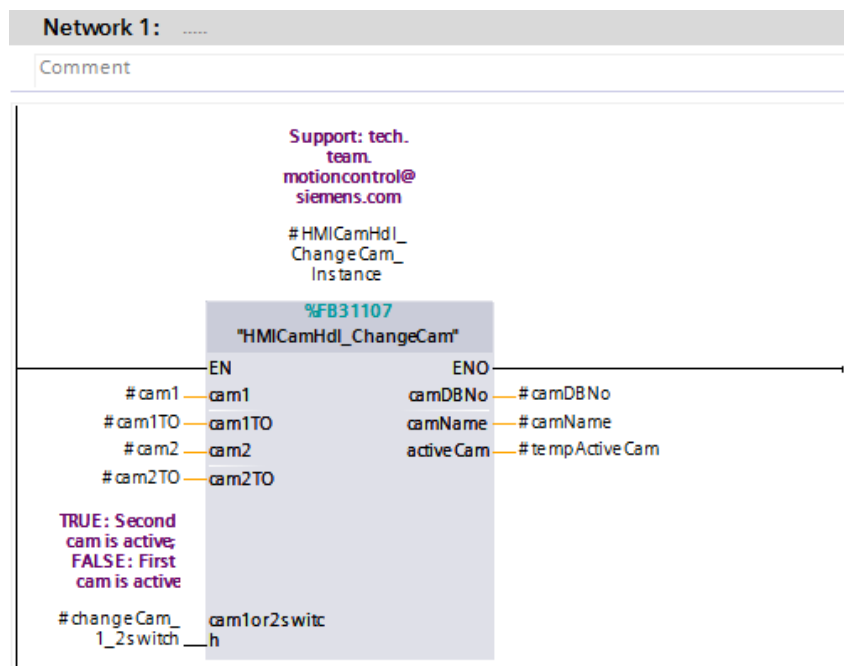
```

# Appendix B

## HMI libraries

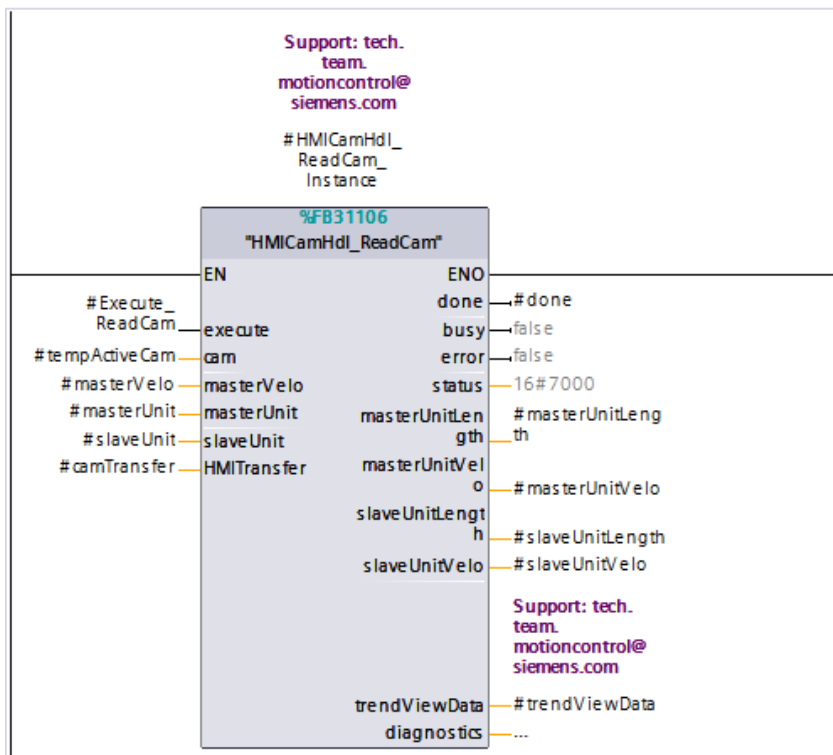
In this appendix, the code used for the implementation of the HMI libraries described in the fourth chapter is reported. It contains both the show cam library and the DB import and export.

### Show cam library



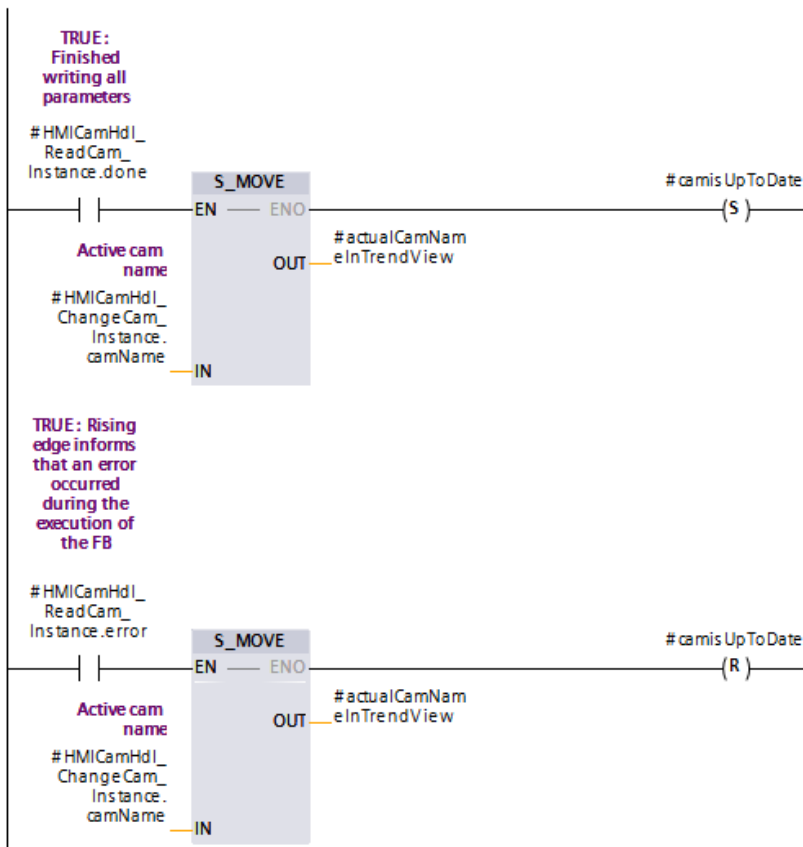
**Network 2:** .....

Comment



**Network 3:** Copy the name of active cam (HMI) and set/reset flag "is up to date"

Comment



---

# DB import and export

## PLC FB

```
#R_TRIG_Execute_Instance(CLK := #Execute, Q => #StartCycle);

CASE #diSelectCase OF
  0:
    IF NOT #Execute THEN
      #Scan_Done := FALSE;
      #Error := FALSE;
    END_IF;

    IF #StartCycle THEN
      IF #WriteDataToHMI THEN
        IF #SysDataTransf THEN
          #db_Num := 10;
          #array_start := 6;
          #element_dim := 2;
          #diSelectCase := 10;
        ELSIF #TimerDataTransf THEN
          #db_Num := 1;
          #array_start := 10;
          #element_dim := 16;
          #diSelectCase := 20;
        ELSE
          #diSelectCase := 30;
        END_IF;

        #attrRetVal := ATTR_DB(REQ := true, DB_NUMBER := #db_Num,
                              DB_LENGTH => #db_Length, ATTRIB => #attByte);
        #array_dim := UDINT_TO_DINT((#db_Length - #array_start) / #element_dim) - 1;
        #cnt_start := 0;
        IF #array_dim > 19 THEN
          #cnt_end := 19;
        ELSE
          #cnt_end := #array_dim;
        END_IF;

      ELSE
        IF #SysDataTransf THEN
          #db_Num := 10;
          #array_start := 6;
          #element_dim := 2;
          #diSelectCase := 60;
        ELSIF #TimerDataTransf THEN
          #db_Num := 1;
          #array_start := 10;
          #element_dim := 16;
          #diSelectCase := 70;
        ELSE
          #diSelectCase := 30;
        END_IF;

        #attrRetVal := ATTR_DB(REQ := true, DB_NUMBER := #db_Num,
                              DB_LENGTH => #db_Length, ATTRIB => #attByte);
        #cnt_end := UDINT_TO_DINT((#db_Length - #array_start) / #element_dim) - 1;
        #cnt_start := 0;

      END_IF;
    END_IF;
  END_IF;
```

```

10: // Write dataSys to HMI
FOR #cnt := #cnt_start TO #cnt_end DO
    #loading_array[#cnt] := "Sys".N[#cnt];
END_FOR;

IF #cnt > #array_dim THEN
    #diSelectCase := 50;
ELSE
    #cnt_start := #cnt;
    #cnt_end := #cnt_start + 19;
    IF #cnt_end > #array_dim THEN
        #cnt_end := #array_dim;
    END_IF;
END_IF;

20:// Write dataTimer to HMI
FOR #cnt := #cnt_start TO #cnt_end DO
    #loading_array[#cnt] := TIME_TO_INT ("Timers".N[#cnt].PT);
END_FOR;

IF #cnt > #array_dim THEN
    #diSelectCase := 50;
ELSE
    #cnt_start := #cnt;
    #cnt_end := #cnt_start + 19;
    IF #cnt_end > #array_dim THEN
        #cnt_end := #array_dim;
    END_IF;
END_IF;

30:
#Error := TRUE;
#diSelectCase := 0;

50:
#Scan_Done := TRUE;
#diSelectCase := 0;

60:// Copy DataSys From HMI
IF #cnt_end <> #array_dim THEN
    #diSelectCase := 30;
ELSE
    IF #array_dim > 19 THEN
        #cnt_end := 19;
    END_IF;
    #diSelectCase := 65;
END_IF;

65:
FOR #cnt := #cnt_start TO #cnt_end DO
    "Sys".N[#cnt] := #loading_array[#cnt];
END_FOR;

IF #cnt > #array_dim THEN
    #diSelectCase := 50;
ELSE
    #cnt_start := #cnt;
    #cnt_end := #cnt_start + 19;
    IF #cnt_end > #array_dim THEN
        #cnt_end := #array_dim;
    END_IF;
END_IF;

70:// Copy DataTimer from HMI
IF #cnt_end <> #array_dim THEN
    #diSelectCase := 30;
ELSE
    IF #array_dim > 19 THEN
        #cnt_end := 19;
    END_IF;
    #diSelectCase := 75;
END_IF;

```

```

75:
FOR #cnt := #cnt_start TO #cnt_end DO
    "Timers".N[#cnt].PT := INT_TO_TIME(#loading_array[#cnt]);
END_FOR;

IF #cnt > #array_dim THEN
    #diSelectCase := 50;
ELSE
    #cnt_start := #cnt;
    #cnt_end := #cnt_start + 19;
    IF #cnt_end > #array_dim THEN
        #cnt_end := #array_dim;
    END_IF;
END_IF;

END_CASE;

```

## HMI write script

```

Sub Sys_Timer_DataWrite(ByVal Sys_Or_Timer, ByRef sSourcePath, ByVal CallFromRead, ByVal bIsPanel)

Dim FileName, VariableName, sUpdateTxtPath
Dim strLine, i, cnt
Dim dActualTime, dMaxCycleTime
Dim Xor_Temp, int1, tag1
Dim oFileSysocE, oFileSysocPC, oFile

SmartTags("Import_Export_Execution") = True
If StrComp(Sys_Or_Timer, "SysData")=0 Then
    SmartTags("Select_Sys_DB") = True
    SmartTags("Select_Timer_DB") = False
    FileName = "\SysData.csv"
    VariableName = "SysData"
ElseIf StrComp(Sys_Or_Timer, "TimerData")=0 Then
    SmartTags("Select_Timer_DB") = True
    SmartTags("Select_Sys_DB") = False
    FileName = "\TimerData.csv"
    VariableName = "TimerData"
Else
    ShowSystemAlarm("Error selecting the File to write")
    Exit Sub
End If

SmartTags("PLC_WriteDataToHMI") = True
SmartTags("PLC_Execute") = True

'Waiting PLC response'
dActualTime = Now()
dMaxCycleTime = DateAdd("s", 5, dActualTime) 'define max time as 5 seconds from now'
Do
Loop While Not (SmartTags("PLC_Done") Or SmartTags("PLC_FB_Error")) And (Now() < dMaxCycleTime)
'Loop Until SmartTags("PLCDone") Or SmartTags("PLC_FB_Error") Or (Now() > dMaxCycleTime)

If Not SmartTags("PLC_Done") And Not SmartTags("PLC_FB_Error") Then
    ShowSystemAlarm("Time expired reading data from PLC")
End If

'Xor'
If StrComp(Sys_Or_Timer, "SysData")=0 Then
    SmartTags("Xor_Mask_SysData") = 59
    For cnt = 0 To SmartTags("PLC_Dimension")
        SmartTags("Xor_Static_SysData")(cnt) = SmartTags("Xor_Mask_SysData") Xor
            SmartTags("PLC_Data_Transfer_array")(cnt)

    Next
    SmartTags("Xor_SysData_Done") = True
ElseIf StrComp(Sys_Or_Timer, "TimerData")=0 Then
    SmartTags("Xor_Mask_TimerData") = 59
    For cnt = 0 To SmartTags("PLC_Dimension")
        SmartTags("Xor_Static_TimerData")(cnt) = SmartTags("Xor_Mask_TimerData") Xor
            SmartTags("PLC_Data_Transfer_array")(cnt)

    Next
    SmartTags("Xor_TimerData_Done") = True
End If

```

---

```

'To end the execution if called from the read script
If CallFromRead = 1 Then
    Exit Sub
End If

'Creazione File e scrittura dati'
If StrComp(bIsPanel,"True")=0 Then
    Set oFileSysoPC = CreateObject ("Scripting.FileSystemObject")
    sUpdateTxtPath = sSourcePath & FileName
    Set oFile = oFileSysoPC.CreateTextFile (sUpdateTxtPath, 1)

    For i=0 To SmartTags("PLC_Dimension")
        strLine = VariableName & ";" & "[" & i & "]" & ";" & SmartTags("PLC_Data_Transfer_array")(i)
        oFile.WriteLine strLine
    Next

Else
    Set oFileSysoCE = CreateObject("filectl.filesystem")
    Set oFile = CreateObject ("FileCtl.File")
    sUpdateTxtPath = sSourcePath & FileName
    oFile.Open sUpdateTxtPath, 2
    For i=0 To SmartTags("PLC_Dimension")
        strLine = VariableName & ";" & "[" & i & "]" & ";" & SmartTags("PLC_Data_Transfer_array")(i)
        oFile.LinePrint strLine
    Next

End If

SmartTags("PLC_Execute") = False
oFile.Close
ShowSystemAlarm ("Data Written on File")
SmartTags("Import_Export_Execution") = False
End Sub

```

---

## HMI read script

```
Sub Sys_Timer_DataRead(ByVal Sys_Or_Timer, ByRef sSourcePath, ByVal bIsPanel)

    Dim FileName, VariableName, sUpdateTxtPath
    Dim strLine, i, cnt, arrItems
    Dim dActualTime, dMaxCycleTime
    Dim Xor_Temp
    Dim oFileSysoCE, oFileSysoPC, oFile
    Dim eof
    Dim ErrorFound_FromFile

    SmartTags("Import_Export_Execution") = True

    If StrComp(Sys_Or_Timer, "SysData")=0 Then
        SmartTags("Select_Sys_DB") = True
        SmartTags("Select_Timer_DB") = False
        FileName = "\SysData.csv"
        VariableName = "SysData"
    ElseIf StrComp(Sys_Or_Timer, "TimerData")=0 Then
        SmartTags("Select_Timer_DB") = True
        SmartTags("Select_Sys_DB") = False
        FileName = "\TimerData.csv"
        VariableName = "TimerData"
    Else
        ShowSystemAlarm("Error selecting the File")
        SmartTags("Import_Export_Execution") = False
        Exit Sub
    End If

    Set oFileSysoPC = CreateObject ("Scripting.FileSystemObject")
    sUpdateTxtPath = "\Storage Card SD" & FileName

    If oFileSysoPC.FileExists(sUpdateTxtPath) Then
        Set oFile = oFileSysoPC.OpenTextFile(sUpdateTxtPath,1)
        cnt = 0
        Do
            strLine = oFile.ReadLine
            arrItems = Split(strLine, ";")
            SmartTags("Temp_array")(cnt) = CInt (arrItems(2))
            cnt = cnt + 1
            eof = oFile.AtEndOfStream
        Loop Until eof

        SmartTags("Temp_dimension") = cnt - 1 'get rid of last increment of while'

        If StrComp(Sys_Or_Timer, "SysData")=0 Then
            'Check if Xor operation was already performed on PLC data'
            If SmartTags("Xor_SysData_Done") = False Then
                Sys_Timer_DataWrite "SysData", SmartTags("L_St_sSDPath"), 1, "True"
            End If
            'Xor execution control'
            For i = 0 To SmartTags("Temp_dimension")
                Xor_Temp = SmartTags("Xor_Mask_SysData") Xor SmartTags("Temp_array")(i)
                If Xor_Temp <> SmartTags("Xor_Static_SysData")(i) Then
                    ErrorFound_FromFile = True
                    i = SmartTags("Temp_dimension") + 1 'Exit immediatly'
                End If
            Next
            'Activation PLC Conditions'
            SmartTags("Select_Sys_DB") = True
            SmartTags("Select_Timer_DB") = False
        End If
    End If
End Sub
```



```

ElseIf StrComp(Sys_Or_Timer,"TimerData")=0 Then
'Check if Xor operation was already performed on PLC data'
If SmartTags("Xor_TimerData_Done") = False Then
Sys_Timer_DataWrite "TimerData", SmartTags("L_St_sSDPath"), 1, "True"
End If
'Xor execution control'
For i = 0 To SmartTags("Temp_dimension")
Xor_Temp = SmartTags("Xor_Mask_TimerData") Xor SmartTags("Temp_array")(i)
If Xor_Temp <> SmartTags("Xor_Static_TimerData")(i) Then
ErrorFound_FromFile = True
i = SmartTags("Temp_dimension") + 1 'Exit immediatly'
End If
Next
'Activation PLC Conditions'
SmartTags("Select_Timer_DB") = True
SmartTags("Select_Sys_DB") = False
End If

'activation of Pop-Up screen if needed'
If ErrorFound_FromFile Then
ActivateScreen "25_PopUp_Continue_Importing", 0
Else
CopyXorValue_To_XorStatic
End If

Else
strLine = "File: " & FileName & " not present"
SmartTags("Import_Export_Execution") = False
ShowSystemAlarm(strLine)
End If

oFile.Close

```

End Sub

```

Sub CopyXorValue_To_XorStatic()
Dim i
If SmartTags("Select_Sys_DB") And Not SmartTags("Select_Timer_DB") Then
For i = 0 To SmartTags("Temp_dimension")
SmartTags("Xor_Static_SysData")(i) = SmartTags("Xor_Mask_SysData") Xor SmartTags("Temp_array")(i)
Next
ElseIf Not SmartTags("Select_Sys_DB") And SmartTags("Select_Timer_DB") Then
For i = 0 To SmartTags("Temp_dimension")
SmartTags("Xor_Static_TimerData")(i) = SmartTags("Xor_Mask_TimerData") Xor SmartTags("Temp_array")(i)
Next
End If
Copy_Data_To_PLC(SmartTags("Temp_dimension"))
End Sub

```

```

Sub Copy_Data_To_PLC(ByRef Dimension)

    Dim i, j, StartValue, EndValue
    Dim dActualTime, dMaxCycleTime
    Dim NumberOfCycles, RemainingElements

    SmartTags("PLC_Dimension") = Dimension
    SmartTags("PLC_WriteDataToHMI") = False
    SmartTags("PLC_Execute") = False

    NumberOfCycles = Dimension \ 50 'gives back the number of times 50'
                                   'is in Dimension without rest'
    RemainingElements = Dimension Mod 50
    For i = 1 To NumberOfCycles
        StartValue = 50*(i-1) 'values are i=1->0; i=2->50; i=3->100; i=4->150; ... '
        EndValue = (50*i)-1 'values are i=1->49; i=2->99; i=3->149; i=4->199; ... '
        For j = StartValue To EndValue
            SmartTags("PLC_Data_Transfer_array")(j) = SmartTags("Temp_array")(j)
        Next
        WaitMs(300) 'wait to clean HMI buffer?? otherwise the forse gives overflow'
                  'error after 99 cycles'
    Next

    For i = j To Dimension
        SmartTags("PLC_Data_Transfer_array")(i) = SmartTags("Temp_array")(i)
    Next
    SmartTags("PLC_Execute") = True

    'Waiting PLC response'
    dActualTime = Now()
    dMaxCycleTime = DateAdd("s", 5, dActualTime) 'define max time as 5 seconds from now'
    Do
    Loop While Not (SmartTags("PLC_Done") Or SmartTags("PLC_FB_Error"))
                And (Now() < dMaxCycleTime)

    If SmartTags("PLC_Done") Then
        ShowSystemAlarm ("Data Imported From File successfully")
    ElseIf SmartTags("PLC_FB_Error") Then
        ShowSystemAlarm("Error occurred while writing DB")
    Else
        ShowSystemAlarm("Time Expired")
    End If

    SmartTags("PLC_Execute") = False
    SmartTags("Import_Export_Execution") = False
End Sub

```

# Bibliography

- [1] ELAU (2008), *PacDrive TPL-FB-ErrorHandler*.
- [2] Schneider Electric, <https://www.se.com/it/it/product-subcategory/2935-servoazionamenti-e-servomotori/>
- [3] Siemens, <https://mall.industry.siemens.com/mall/en/WW/Catalog/Products/10239894>
- [4] Siemens (2017), *SIMATIC S7-1500 and S7-1500T Motion Control V4.0 in TIA Portal V15*.
- [5] Siemens (2017), *Creation of cam disks at runtime for S7-1500T*. [*Library Advanced cam creation*].
- [6] Siemens (2017), *Creation of cam disks at runtime for S7-1500T*. [*Library LCamHdl - Cam creation based on XY points*].
- [7] Siemens (2020), *Creation of cam disks at runtime for S7-1500T*. [*Library LCamHdl - Basic cam creation*].
- [8] Siemens (2020) *Configuring a SINAMICS S120 with Startdrive*.
- [9] Siemens, (2020) *SINUMERIK, Parameter description*.
- [10] Siemens, (2014) *SINAMICS G/S: HMI direct access*. [*SINAMICS G120 SINAMICS S120*].
- [11] Siemens, (2017) *Axis Control Blocks for S7-1500 / S7-1500T*. [*LAxisCtrl for SIMATIC*].