

POLITECNICO DI TORINO

Master of Science in Mechatronic Engineering

Master degree Thesis

Analysis and developement of a semi-automatic assembly machine



Supervisors:
Luigi MAZZA
Terenziano RAPARELLI

Candidate:
Luca TURTÙ

October 2018

Acknowledgments

Vorrei iniziare ringraziando la mia famiglia, in particolar modo i miei genitori.

Senza i vostri sforzi e sacrifici non sarei arrivato dove sono, quindi ci tengo a ringraziarvi di cuore e a dirvi che questo risultato è soprattutto vostro e questa tesi è a voi dedicata.

Vorrei poi ringraziare di cuore mio fratello Giorgio, che mi ha sempre sostenuto ed aiutato, stimolandomi con la tenacia e la perseveranza che lo contraddistinguono.

Un ringraziamento speciale a Greta, la mia più grande forza, che mi ha aiutato e soprattutto sopportato nei momenti in cui ne avevo più bisogno. Grazie per essermi sempre stata vicina.

Come non ringraziare i miei amici, in primis Giovanni, che riesce sempre a capirmi quando gli altri non riescono e che ormai considero più un fratello che un amico, e Lorenzo, anche lui un amico come ce ne sono pochi, paziente e disposto a darti una mano, anche a costo di rimetterci in prima persona.

Un ringraziamento anche a Mirco, Genny, Giulia, Luca, Gloria, Francesco, Melissa e Arianna perchè siete degli amici eccezionali e spero che sarete lì a sopportarmi anche in futuro.

Ci tenevo poi a ringraziare tutti i miei ex compagni di liceo e di università, i miei coinquilini, sia vecchi che nuovi, con cui ho passato delle bellissime esperienze ed avventure, perciò un grazie va anche a Martino, Roberto, Paolo, Marco e Andrea.

Un grande ringraziamento va anche ai miei due relatori, sempre pronti a rispondere ad ogni mio dubbio o perplessità, ed in particolar modo a Sandro e Alessio per avermi accolto nella loro azienda come fossi uno di loro.

Infine volevo ringraziare Roberto, Carlo, Stefano, Andrea, Matteo e Quinto che mi hanno seguito durante tutto il percorso, per avermi sopportato e aiutato a crescere.

Grazie a tutti!

Table of contents

| | |
|---|-----------|
| Acknowledgments | I |
| 1 Introduction | 1 |
| 1.1 Products assembled | 2 |
| 1.2 Unit description | 3 |
| 1.3 PLC Programming | 6 |
| 1.4 Program Blocks | 7 |
| 2 Hardware Configuration | 10 |
| 2.1 Decentralized Systems | 10 |
| 2.2 Motors | 14 |
| 2.3 Robots | 18 |
| 2.4 Other Devices | 21 |
| 2.4.1 Press | 21 |
| 2.4.2 Measurement Instruments | 21 |
| 2.4.3 Laser and Dataman | 22 |
| 3 Operation 150 | 23 |
| 3.1 Dowel Pin Insertion | 23 |
| 3.1.1 Cycle Description | 28 |
| 3.1.2 Feeder Management | 31 |
| 3.2 Runout Test | 32 |
| 4 Operations 160 and 170 | 40 |
| 4.1 Operation 160 | 41 |
| 4.1.1 EOL Test | 41 |
| 4.1.1.1 Test cycle | 45 |
| 4.1.2 Leak Test | 48 |
| 4.2 Operation 170 | 51 |

| | | |
|----------|---|-----------|
| 5 | Software Implementations | 54 |
| 5.1 | Safety Management | 54 |
| 5.2 | Motors | 57 |
| 5.2.1 | General Management | 58 |
| 5.2.2 | Absolute and Relative Positioning | 62 |
| 5.3 | Robots Management | 64 |
| 5.3.1 | General Function Block | 64 |
| 5.3.2 | Robot Programming | 66 |
| 5.4 | Piece and Pallet Identification | 73 |
| 5.4.1 | Data Blocks Used | 73 |
| 5.4.2 | Informations Transfer | 76 |
| 5.5 | SCADA System | 81 |
| 6 | Conclusions | 89 |
| | Bibliography | 91 |

List of tables

| | | |
|-----|--|----|
| 4.1 | Viscosity Coefficient of Gases and Liquids. [13] | 49 |
| 5.1 | Pallet DB. | 73 |
| 5.2 | Process DB. | 75 |
| 5.3 | Configuration DB. | 76 |

List of figures

| | | |
|------|--|----|
| 1.1 | Layout of the last unit. | 5 |
| 1.2 | Ladder Diagram [5]. | 6 |
| 2.1 | ET 200SP - 7300AC1 configuration. | 11 |
| 2.2 | CPX-CP - Configuration. | 13 |
| 2.3 | Connection between valve manifolds/input modules and CPX-CP system. | 13 |
| 2.4 | Drive - Status Word. | 15 |
| 2.5 | Drive - Control Word. | 17 |
| 2.6 | Robot - Input Addresses. | 19 |
| 2.7 | Robot - Output Addresses. | 20 |
| 3.1 | CPX-CP System - Zoom on Dowel Pin Block Valve and Module Input. | 23 |
| 3.2 | Kistler Press - Dowel Pin Insertion. | 24 |
| 3.3 | Indirect Command Signals | 27 |
| 3.4 | Dowel Pin Feeder. | 31 |
| 3.5 | Circular Runout. [10] | 32 |
| 3.6 | Concentricity. [10] | 33 |
| 3.7 | Runout Station - Transducers. | 34 |
| 3.8 | TQM Data Block. | 36 |
| 3.9 | Runout Station. | 37 |
| 3.10 | Runout in polar (above) and cartesian coordinates (below). | 39 |
| 4.1 | EOL Test Stations. | 43 |
| 4.2 | Air Circuit EOL Test. | 44 |
| 4.3 | Leak Test Stations. | 50 |
| 4.4 | Laser Station. | 52 |
| 5.1 | Safety Doors Function. | 55 |
| 5.2 | FC8002 - Safety Doors Function compiled. | 57 |
| 5.3 | Transfer Motor Input and Output Data. | 59 |
| 5.4 | Motor Data Block - Input and Output data. | 60 |
| 5.5 | Drive - Operating Mode. | 61 |

| | | |
|------|---|----|
| 5.6 | Target Position Transfer. | 62 |
| 5.7 | Motor Following Error. | 63 |
| 5.8 | Robot Function Block - Commands and Faults. | 66 |
| 5.9 | Pallet DB - Data Block Structure. | 74 |
| 5.10 | Pallet block - Reading the antenna. | 77 |
| 5.11 | Read Pallet Tag Function. | 78 |
| 5.12 | Supervision Layout. | 85 |
| 5.13 | Motor Parameters Settings. | 86 |
| 5.14 | Supervision - normal configuration above, test-skip configuration below | 88 |

Chapter 1

Introduction

Nowadays the assembly machines are widely used in the industries, since they reduce the production time and human error, while increase the accuracy and repeatability.

This thesis concerns, precisely, the analysis and development of the last part of a semi-automatic machine, that assembles and then tests two different camshaft phaser (intake and exhaust) for automotive applications. The line was completely designed and built by OTS Assembly, which is a company that deals, precisely, with the construction of prototypes of assembly machines.

The machine is a prototype of a pallettized assembly line, that means that the various components assembled to form the piece are processed directly on the pallet while this one moves along the line.

The pallet, in this case, is a rectangular part (250x200), designed to contain the piece to be worked and any components to be inserted, moved by two parallel belts running on aluminum profiles.

The machine produces two different pieces, that differ in functionalities, specifications, dimensions and especially for the operations they must undergo. Thus, two different sets of 60 pallets are needed.

This means that there are 60 pallets moving along the line.

Since there are a lot of operations to do to assemble and test the products, to better organize the work the machine has been divided in different units, each of them grouping different operations to be applied. There are 7 units, for a total of 22 operations (11 for intake and 11 for exhaust) distributed among them, operations that vary depending on the tipology selected.

Moreover the machine can work one tipology at a time, that must be specified before starting. This is because the assembly line requires mechanical retooling of the various stations, in addition to the changing of the pallet type.

The main purpose was, first of all, programming, and subsequently control, all the devices present in the last unit, such as robots, drives, measurement instruments, sensors, actuators and so on, so that it is possible to create operating cycles of the stations that have to process the pieces. All these devices are managed by a *fail-safe PLC Siemens 1500F (CPU 1517F-3 PN/DP)*, that is a controller that implements machine safety in addition to managing the normal operations of the machine itself. Then creating a supervision program to allow the operator that uses the machine to be able to interface with it.

1.1 Products assembled

The assembly line produces cam phasers, both **exhaust** and **intake**, which are used to adjust hydraulically the position respectively of the exhaust and intake camshafts while the engine is operating.

The cam lobes on the camshafts are meant to control the timing of the opening (or closing) of the poppet valves and therefore the timing and quantity of air inserted into the engine or the gases exhausted from the engine: the entire cam can be rotated clockwise or counterclockwise in relation to the crankshaft position to make the poppet valves open sooner or later in the piston stroke.

The main benefits attainable by using such device can be related to [2]:

- **reduced pumped work**, due to the possibility to avoid throttling, managing the intake air quantity directly through the intake valve opening duration, timing and maximum lift
- **air motion control**, thanks to a Late Intake Valve Opening (the intake flow velocity is higher since the piston velocity is higher)
- **volumetric efficiency optimization**, taking advantage of fluid mass inertia

by means of Late Intake Valve Closing, Early Exhaust Valve Opening and increased valve overlap.

This usually brings to an increase of maximum torque (the increased volumetric efficiency can be considered as “natural supercharging”). These strategies are effective at medium-high engine speed

The cam phasers have two main elements, a stator and a rotor.

The stator is composed of a sprocket (or housing) fastened between an inner plate and an outer plate. The outer plate is directly bolted to the camshaft, so that a chain (directly hooked to the sprocket) transmit motion from the crankshaft to the camshaft.

The rotor, instead, positioned inside the stator, is connected to the camshaft using a dowel pin and they can move independently thanks to a backlash, in general set to be about 20° .

Thus the rotor is responsible of driving the camshaft.

1.2 Unit description

Each unit has been considered as a machine of its own, that can work regardless of the others, as long as no emergency occurs.

There are safety guards to protect the operator, that are used to open safety relays in case one of them opens. If this happens, the entire line does not stop, but only the unit in which one of these opens.

When a safety guard is opened, the air is discharged, so everything powered by pressurized air fails. When the guard is closed, the machine needs a ‘restart’ command to come back working again, done by pressing the special restart button on the unit pushbutton.

The same thing happens when other critical situations happen, for instance when a circuit breaker present in the cabinet of the unit opens. When one of the emergency buttons along all the line is pressed or for instance a circuit breaker inside the general cabinet opens, instead, all the assembly line stops. To restore the emergency it is necessary to release the red emergency buttons pressed or close back the circuit

breaker and then to press a ‘master start’ button, that is present only at the beginning of the line on the general pushbutton.

Moreover, each unit has its own operation to be performed. For what concerns the last unit, showed in (Fig 1.1) there are three operations, which can be different or common depending on whether they have to be applied to an intake piece, an exhaust one, or both, and that will be explained in details in the following chapters. Two robots are present. Each of them has two *ad hoc* grippers designed to permit a form-fit pick, in order to have a reduced cycle time.

The robots are supposed to pick the piece directly from the pallet and place it under the station where it must be worked, and vice-versa, once the station finished, to put it back on the pallet.

- **OP. 150**

The first operation is different for the two products.

- EXHAUST: a press inserts a dowel pin into the exhaust cam phaser, whose function is to connect the rotor of the cam phaser directly to the camshaft
- INTAKE: a *Runout* measurement is performed to check how much one entire feature or surface varies with respect to a datum when the part is rotated 360° around the datum axis.

There are two stations, that cannot run simultaneously though, since the software used (**ITAGEO 6** from TQMitaca) can manage only one at a time.

- **OP. 160**

This operation is common to both products and consists of two tests.

- The first one is a End-Of-Line (*EOL*) test for Automotive Applications that checks the effective functionality of the finished product by simulating some relevant working conditions and measuring the responses.
- The second one is a *Leak Test*. As the word says, this test is meant to check that there are no fluid-losses.

In this case, instead, there are respectively 2 stations for the EOL test and 4 for the Leak test, that can work simultaneously, given the lenght of the operations.

- **OP.170**

Also this one is common to both and consists of marking the good products coming out of the tests with a laser, while rejecting the bad ones.

Before starting with the description of the operations, it is important to illustrate what components are present in the unit and how they have been configured inside the **SIEMENS TIA Portal**, an environment for creating the program running the assembly line.

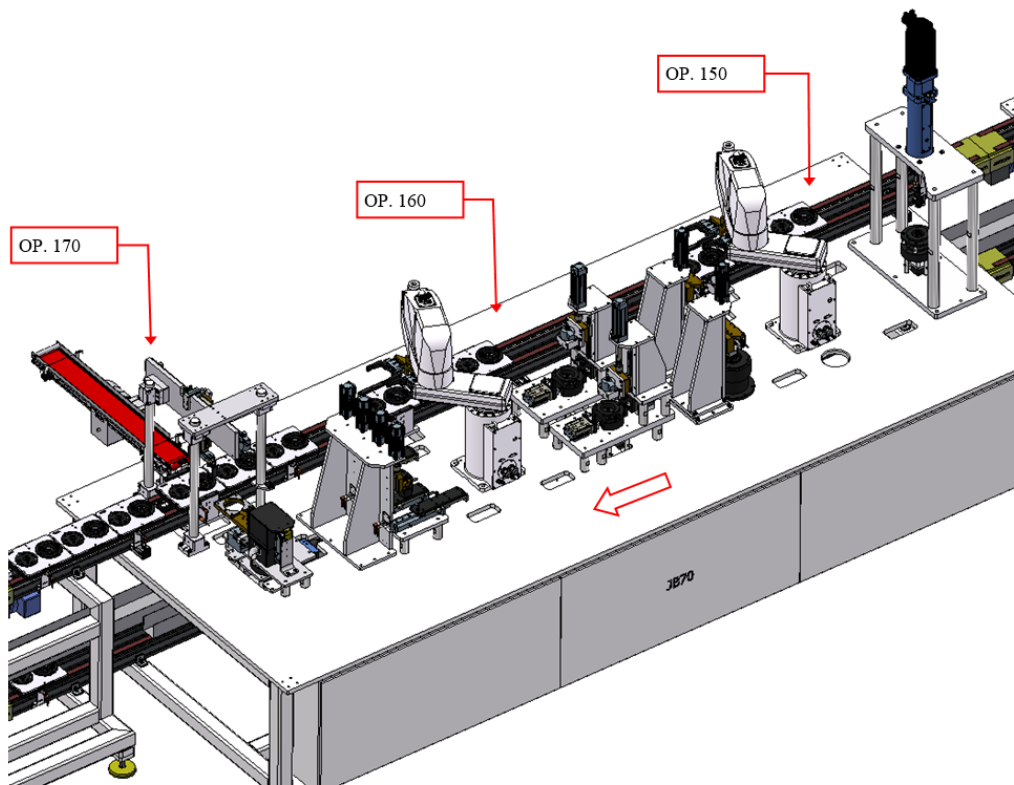


Figure 1.1: Layout of the last unit.

1.3 PLC Programming

The entire machine is controlled by a programmable logic controller (PLC), that is *a real-time system optimized for use in severe conditions such as high/low temperatures or an environment with excessive electrical noise. This control technology is designed to have multiple interfaces (I/Os) to connect and control multiple mechatronic devices such as sensors and actuators.* [3]

This programmable logic controller is programmed through specific program languages, that can be both **graphical** and **textual**, depending on the application. Between the *graphical* languages, one of the most known is the so called **Ladder diagram**.

This particular programming way makes use of a particular logic that is similar to relay schematic circuits.

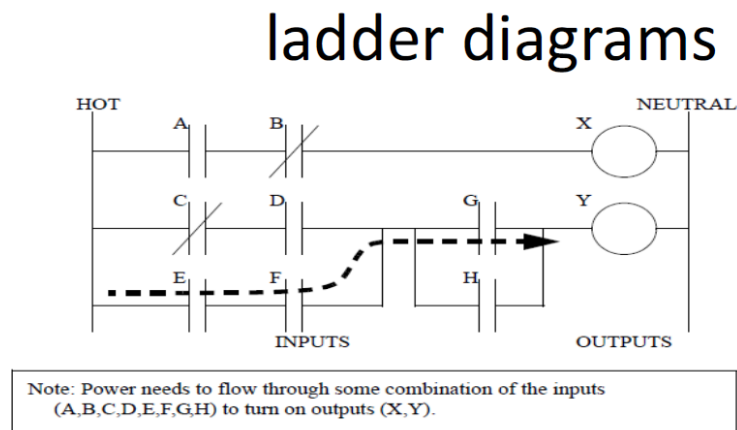


Figure 1.2: Ladder Diagram [5].

The fundamental elements of a ladder are the so called “**Contacts**” and “**Coils**”, that are often associate with the inputs and outputs.

An important observation to make is that contacts can be tied either to inputs or outputs, while coil are generally associated to outputs.

The two vertical lines are the so called contacts, while the circles represent the coils. Depending on the way the contacts are opened or closed, the power can flow from

the hot rail (left) to power the coils, and finally to the neutral rail.

Power flows through these contacts when they are closed and this depends on the contact type, that can be:

- *Normally Open (NO)*: this contact is closed when the input or output (associated to it) status bit is 1. A graphical example is represented by the *A* contact in Fig. 1.2
- *Normally Closed (NC)*: this contact is closed, instead, when the input or output status bit is 0. A graphical example is the *B* contact in the picture.

Moreover, contacts can be also associated to timers and counters to perform particular operations. An example of how timers can be implemented is showed in chapter 3 (3.1).

Siemens has implemented its own Ladder, called **KOP** and that will be present later on, since largely used and that implements other functionalities.

Two others languages worth being briefly explained, that are **Instruction List** and **Structured Text** and that are part of the *Textual* languages.

The first one is similar to assembly and it's a low programming language, while the second is more similar to modern programming language and it's a high programming language.

Also in this case Siemens implemented its own languages, that are respectively **AWL** and **SCL**, that have both been used and explained later on.

1.4 Program Blocks

Program blocks are the instruments necessary for programming any kind of software for the Siemens PLC used.

They can be of different types:

- **Data block** is a particular block that permits to save program variables and data

- **Function Block** which is a code block that can store values directly inside its own instance data block
- **Function** is a code block or a subroutin but without a dedicated memory

Several DBs are necessary to store informations. In particular there are some of them necessary exclusively to exchange informations with the outside.

They can be *Optimized* or not, depending on the use they have to perform: if they are optimized, each DB element present on it has a symbolic address, that is adjusted automatically if other elements are added, so it is not possible to point it directly with an address. This is not always a good solution, especially for those situations in which it is important to know which address to point, where *non Optimized* DBs are used.

To create a DB it is only necessary to specify a name and a number, that must be univocal.

An example can be the DB intended for storing informations coming by the motor (or to send to the motor), for the TQM software, or for other applications.

Through a **SCADA** system, implemented in Visual Basic (see 5.5), it is possible to control, for example, the position of the motors or the actuators, feedbacks of the alarms and to pass whatsoever information necessary for the operator to use the machine, such as the number of products to produce, the type of piece, particular parameters for the tests and so on.

This supervision program can read and write determinated data blocks created inside the PLC.

These DBs are only used to exchange informations, that are then passed to other DBs specific for the device that are necessary for the implementation of the PLC software.

The informations sent to those DBs are then passed to other DBs, specific for the device to be controlled to store particular data to be passed to them or received by them, such as input and output data.

Infact, especially for the output signals, direct commands are not used, both because it is possible that the command remains set and because the same output can be used in different situations.

For example it is possible to command an actuator in automatic mode, during the normal operation of the machine, or in manual mode, for example during calibration operations.

Both these commands are passed through the same signal output, so it is necessary to distinguish between the two situations.

There are also other blocks to be added into the list, for instance the **Program Block**, which is the main block of the program, used only once and called cyclically. Moreover, in a similar way to any other programming language, it is also possible to recall functions within other functions or function blocks.

Infact, the Program Block, mentioned before, contains other sub-mains program, one for each unit, which, in turn, contain functions to check and control all the sensors or processes.

Chapter 2

Hardware Configuration

First of all it was necessary to specify the controller used, that is a **SIMATIC S7-1500** with a fail-safe **CPU 1517F-3 PN/DP**. This means that one CPU implements machine safety (through emergency buttons, safety guards and so on) in addition to managing the normal operations of the machine itself.

Given the dimensions of the assembly line, the inputs/outputs can be very far apart from the CPU. To avoid wiring problems several **SIMATIC ET 200SP** nodes and **FESTO CPX-CP** systems have been used.

ET 200SP is a multi-functional distributed I/O system that comprises an interface module to communicate with the CPU via PROFINET, up to 64 I/O modules plugged into passive base units and a server module, which completes the structure [6].

Also the CPX-CP is a distributed I/O system that communicates via PROFINET with the CPU. The difference is in the communication with the modules connected to it, that takes place through another fieldbus instead (Ethernet Powerlink). This system is responsible for controlling actuators and sensors.

In the next section of this chapter it is described how these distributed I/O systems have been configured inside the TIA Portal development environment and how they interact with the other elements present in the line so that the PLC can communicate with them all.

2.1 Decentralized Systems

First of all it is important to specify that, to make the PLC communicate with every device in its network, a GSD file is required. A GSD (General Station Description)

file contains all the informations necessary to describe a device, such as the communications options and the available diagnostic. The communication takes place via PROFINET, that is an industrial Ethernet, so every device must have its own IP address, to be uniquely identified, as well as a name.

Once chosen the appropriate GSD, to the ET 200SP has been given an IP address and the name 7300AC1. In the next figure(Fig 2.1) is represented in details how this system is composed.

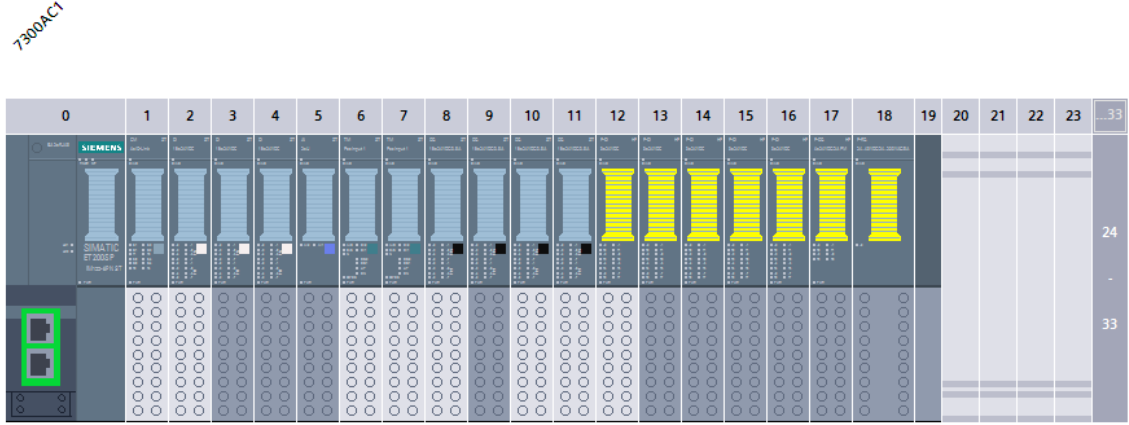


Figure 2.1: ET 200SP - 7300AC1 configuration.

To each module in the figure has been assigned a starting Input/Output Address, so that each signal managed by its own module can be associated to a particular subaddress, in order to be used to command outputs or to receive input feedbacks.

Module **0** is the interface module that communicates directly with the CPU, as mentioned before, while module **19**, is the server module, a mechanical/electrical device that closes off the backplane bus of the ET 200SP.

All the other elements present are I/O interface modules: the light grey units are supplied with 24 V, while the dark ones are fed by the left module.

In particular these modules are:

- **1**: 4-channel serial interface module with IO link devices, used to connect three antennas, one for each operation, that read a pallet tag present on each pallet (see 5.4)

- **2-4:** modules with 16 digital input (each) at 24V, where are connected feedback relay signals coming from safety guards and emergency buttons, as well as all the others pushbutton used by the operators to interact with the machine, such as start cycle button, stop cycle, jog movement, automatic movement and so on
- **5:** analog input module 16-bit, used to connect torque transducers to measure the torque applied to the pieces during EOL test
- **6-7 :** technology modules (TM) PosInput, to whom are connected two absolute encoders, also used during EOL test, to check the position of the pieces
- **8-11:** modules with 16 digital output (each) at 24V, used to give commands such as ‘*Start/Stop*’, ‘*Reset*’ through the pushbuttons, to command belt conveyors and the laser
- **12-16: fail-safe** modules with 8 digital input (each) at 24V, to whom are connected all the elements necessary to guarantee machine safety, so all the emergency pushbuttons and safety guards present on the unit (see [5.1](#))
- **17: fail-safe** module with 4 digital output 24V to command the restoration of the air pressure inside the machine, that goes down whenever a safety guard is opened or an emergency pushbutton pressed

For what concerns the **CPX-CP** system, it provides decentralized control (since in high-speed machines short cycle times and short pneumatic tubing are required) of the actuators and sensors present in the unit through a CPX terminal, that communicates via PROFINET with the CPU, and CP interface modules. Also in this case there is only one CPX-CP system (Fig. [2.2](#)) for the last unit. An IP address has been given to the CPX, as well as a name, 7400AC1, and the same for the two CP interface modules. The first one, named 7400C1, handles 16 Bytes Input/ 16 Bytes Output, while the second one, named 7400C2, 4 Bytes Input/ 4 Bytes Output. [\[7\]](#)

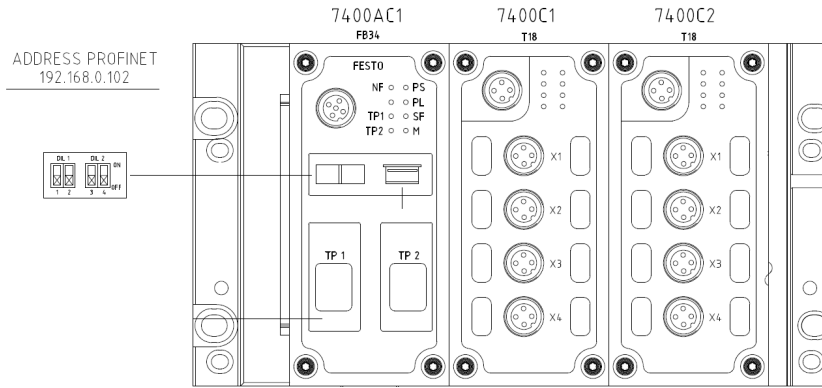


Figure 2.2: CPX-CP - Configuration.

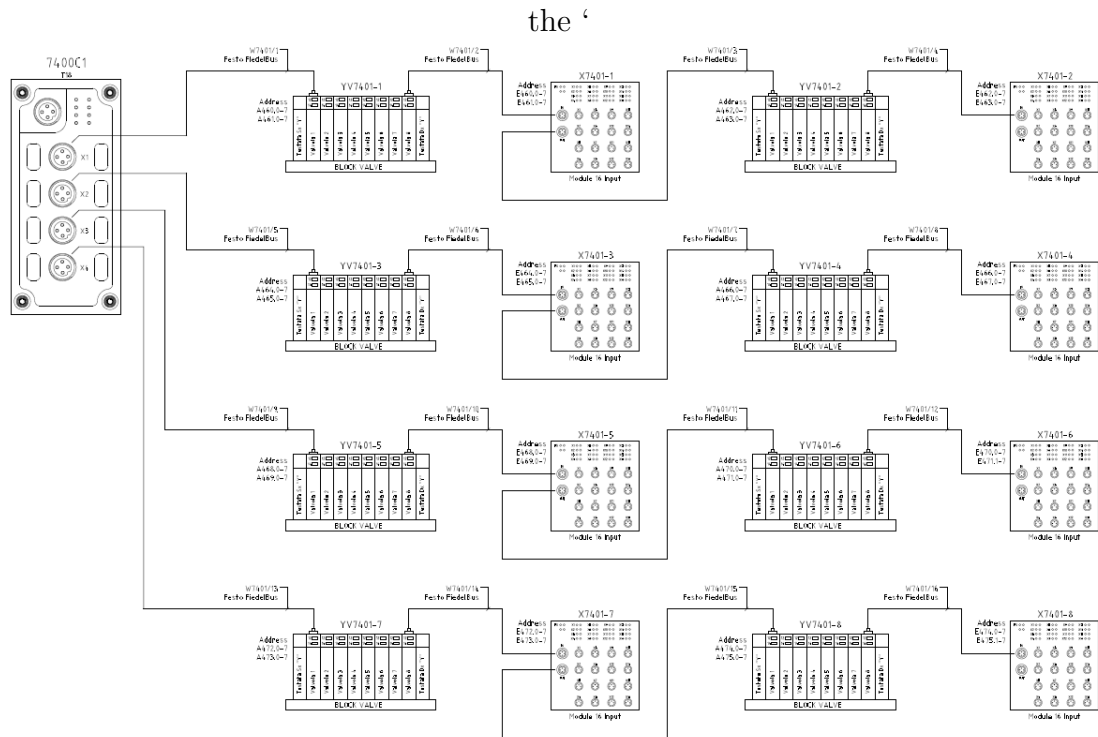


Figure 2.3: Connection between valve manifolds/input modules and CPX-CP system.

The CPI modules are directly connected to FESTO valve manifolds, responsible to command the valves to move the actuators, and to input modules, used to check sensors instead. An example is showed in the following picture (Fig. 2.3), that describes the connection between the elements just described.

2.2 Motors

In addition to the actuators, there are also five synchronous BOSCH motors with absolute encoder in the unit.

To control and interface them with the PROFINET fieldbus, BOSCH INDRA-Drives are required.

As happened for the other devices, also for the drives a GSD file has been inserted so that it was possible to establish what informations share between the drive and the PLC, as well as a starting Input/Output Address and an IP address.

All five motors have been interfaced through a 6 Input Words (for the PLC) and 11 Output Words (from the PLC) communication telegram.

The **6 Input Words** are required to pass to the fieldbus master, that is the PLC, informations regarding the status of the drive or some important parameters of the motor that the drive is controlling.

First of all it was necessary to configure the Drive, to specify how to control it. So the ‘operation mode’ of the drive has been set to ‘Drive-Internal Interpolation’, in such a way that the drive toggles a specific bit when the motor position is in a certain window, given by the difference between the required position the motor has to reach and the actual position read by the encoder.

Three others secondary operations modes had to be set in order for the communication to work, as showed in (Fig. 2.4, bit 8/9).

All the other bits present in the image compose the STATUS word of the drive, that tells the PLC whatsoever information required for a correct use of the motor.

For example with the bit number 2 the drive tells that the axis of the motor is still, or bit 4 that the required position has been reached.

The STATUS word is one one word out of the six, that communicate the following informations instead:

| Bit | Designation | Bit Combination |
|-------|---|--|
| 0/1 | Operating Mode Acknowledgment | 10 : operating mode 01 : no longer relevant 00 : parameter mode |
| 2 | In reference (status of reference encoder) | 0 : relative 1 : homed |
| 3 | In standstill | 1 : If the actual velocity is inside a certain window (aproximated to 0) |
| 4 | In Position | The command value reached for drive-internal interpolation: 1 : if $ (Target\ Position - Feedback\ Position) $ is in a certain window. |
| 5 | Command Change Bit | 1 : if command status has changed 0 : if command status has not changed |
| 6 | Operating Mode Error | 1 : error in transition command 0 : no error in transition command |
| 7 | Status of Command Value Processing / Drive Halt | 1 : drive does not follow command value input (Drive Halt is active) 0 : drive follows command value input (Drive Halt is off) |
| 8/9 | Actual Operating Mode | 00 : primary operation mode, chosen to be 'Drive-Internal Interpolation' 01 : secondary oper. mode 1 , 'Drive-Controlled positioning' 10 : secondary oper. mode 2, 'Velocity Control' 11 : secondary oper. mode 3, 'Velocity Control' |
| 10 | Command value Acknowledgment | By toggling the bit the drive acknowledges the acceptance of the "positioning command value" |
| 11 | Diagnostic Message | The bit is set if a class 3 diagnostics message is present. |
| 12 | Warning Message | The bit is set if a class 2 diagnostics warning is present. |
| 13 | Error Message | The bit is set if a class 1 diagnostics error is present (drive interlock). |
| 14/15 | Ready For Operation 1/2 | 00 : not ready to power on 01 : ready for power on 10 : control and power sections ready for operation 11 : in operation, with torque |

Figure 2.4: Drive - Status Word.

- **Actual Position:** double integer data (DInt)
- **Actual Speed:** double integer data (DInt)
- **Actual Torque:** integer data (Int)

The data types, i.e. DInt or Int, have been chosen this way because these are the formats with which the drive exchanges these kind of informations.

A similar description can be made for the **11 Output Words** required to send data from the PLC to the drive, so that it is possible to command the motor depending on the application.

The first Word sends commands through individual bits (Fig. 2.5).

As an example the combination of bits 6 and 7 is responsible for the way the motor turns: if both bits are equal to 0 the motor is set to positioning active, that means that the motor goes to the target position set by the PLC. If the combination of bits is 01 or 10 the motor moves in ‘Jog’ mode, i.e. the continuous movement of the motor in a specific direction, while it is 11 the motor is stopped. The required position, mentioned before, together with other data need to be told to the drive and that is the reason of the other words.

- **Target position:** sends a double integer data (DInt) to change the position the motor has to reach
- **Speed:** a double integer (Dint) to set the required operational speed (depending on the situation, the motor needs to receive an automatic cycle speed for the normal working conditions, or a jog speed, required for particular situations, for instance calibration operations, normally up to ten times slower than the automatic one)
- **Acceleration and Deceleration:** two double integer data (DInt), normally set to the maximum value
- **Torque Limit Positive and Negative:** two integer data (Int), to pass values of the maximum and minimum torque reachable from the motor

| Bit | Designation | Bit Combination |
|-------|-----------------------------------|---|
| 0 | Command Value Acceptance | |
| 1 | Operating Mode Setting | 0 -> 1 : change to operating mode 1 -> 0 : change to parameter mode |
| 2 | Going to zero (Homing Position) | 0 -> 1 : start homing command 1 -> 0 : complete homing command |
| 3 | Absolute/Relative Target Position | 0 : positioning command value is processed as absolute target position 1 : positioning command value is processed as relative travel distance |
| 4 | Immediate Block Change | 0 : Positioning command value is only accepted after the last active target position was reached 1 : positioning command value is immediately accepted upon toggling of command value acceptance |
| 5 | Clear Error | 0 -> 1 : start command clear error 1 -> 0 : complete command clear error |
| 6/7 | Positioning / Jogging | 00 : positioning active (automatic move) 01 : infinite travel in positive direction (Jog+) 10 : infinite travel in positive direction (Jog-) 11 : stopping the axis |
| 8/9 | Command Operating Mode | 00 : primary operation mode, choosen to be 'Drive-Internal Interpolation' 01 : secondary oper. mode 1 , 'Drive-Controlled positioning' 10 : secondary oper. mode 2, 'Velocity Control' 11 : secondary oper. mode 3, 'Velocity Control' |
| 10/11 | Reserved | |
| 12 | IPOSYNC | Interpolator clock (only in cycl.pos. control): toggles when new command values are transmitted |
| 13 | Drive Halt | 0 -> 1 : drive start 1 -> 0 : Drive Halt is on, i.e. the drive is shut down immediately |
| 14 | Drive Enable/ Reserved | |
| 15 | Drive ON | 0 -> 1 : drive enable 1 -> 0 : best possible deceleration |

Figure 2.5: Drive - Control Word.

In the next chapters it is described how these Status and Command words have been utilized.

2.3 Robots

To take the pieces from the pallet and place them under the work station, two SCARA robots **EPSON G-10** have been used.

Also in this case a controller is required to interface the robots with the PLC so that it is possible to control them and receive feedbacks.

The communication telegram between the PLC and the controller is composed, this time, of 32 Bytes Input and 32 Bytes Output, which is one of the default fieldbus configurations for the controller RC-700A used.

The association between the address and the corresponding command (or feedback), however, has been changed with respect to the default configuration, depending on the operation the robot is involved in. This means that the robot present on OP 150 has a different address configuration with respect to the robot on OP 160).

Moreover most of the bytes have not been used, but provided as reserves for any future addition.

The first byte, similar to the motor drive, is composed of individual bits to give standard commands to the robot or to receive feedback informations from it.

In the next picture (Fig. 2.6) are showed the Input addresses for the PLC and the elements directly connected with the robots (i.e. the grippers), that are Outputs for the controller named 7250AZ1 of the robot working on OP 150. The addresses not present in the table have not been assigned.

The same thing has been done for the Outputs, that this time are Inputs for the PLC (Fig. 2.7). Both Input and Output addresses go from 512 to 767.

A clarification must be done: the addresses showed in the pictures are the one configured inside the robot configuration program. Once established the correct GSD a starting Input/Output address has been choosen equal to 4250.

The robot couples automatically all its addresses with the ones set in the PLC starting with the first one.

| Name | Robot Input Address | Functionality |
|------------------------------|------------------------|--|
| Start | 512 | Bit to start the robot |
| Stop | 513 | Bit to stop the robot |
| Pause | 514 | Bit to pause the robot |
| Continue | 515 | Bit to restart the robot from pause |
| Reset | 516 | Bit to reset eventual errors or collisions |
| Motor_On | 517 | Bit to turn on the motor |
| Motor_Off | 518 | Bit to turn off the motor |
| Home | 519 | Bit to reach the set home position |
| Safe_Pos_Dowel_Pin_Press | 536 | Bit that tells the controller that the robot is free to put a piece under the press station |
| Safe_Pos_Runout_1 | 537 | Bit that tells the controller that the robot has the necessary security to put the piece under the first Runout station |
| Safe_Pos_Runout_2 | 538 | Bit that tells the controller that the robot has the necessary security to put the piece under the second Runout station |
| Safe_Pos_Index_Pallet | 539 | Bit that tells the controller that the robot has the necessary security to put/pick the piece from a pallet |
| Free_Station_Dowel_Pin_Press | 544 | Bit that tells the controller that the the press station is free |
| Free_Station_Runout_1 | 546 | Bit that tells the controller that the first Runout station is free |
| Free_Station_Runout_2 | 547 | Bit that tells the controller that the second Runout station is free |
| Free_Station_Index_Pallet | 532 | Bit that tells the controller that the pallet is free |
| Speed | DInt starting from 560 | Double integer data to set the required speed |
| Mission_Code | DInt_starting from 576 | Double integer data to communicate the mission code associated to an action the robot has to perform |
| Product_Code | DInt starting from 592 | Double integer data to communicate the piece the robot is manipulating |

Figure 2.6: Robot - Input Addresses.

| Name | Robot Output Address | Functionality |
|--------------------------|------------------------|---|
| Ready | 512 | Bit that tells the robot is ready |
| Running | 513 | Bit that tells the robot is running |
| In_Pause | 514 | Bit that tells the robot is in pause |
| Error | 515 | Bit that tells the robot is in error |
| E_Stop_On | 516 | Bit that tells that robot is stopped for an emergency occurrence |
| Safety_Guard_On | 517 | Bit that tells when a safety guard opens |
| System_Error | 518 | Bit that tells there is a system error |
| Warning | 519 | Bit to advice there is a warning |
| Motor_On | 520 | Bit that tells the motor has been turned on |
| At_Home | 521 | Bit that tells when the home position has been reached |
| Auto_Mode | 522 | Bit that tells the auto mode has been set |
| Open_Gripper_1_Command | 528 | Bit to command the opening of the first of the two grippers present on the robot tool |
| Close_Gripper_1_Command | 529 | Bit to command the closing of the first gripper |
| Gripper_1_Open | 530 | Bit that tells that the first gripper has been opened |
| Gripper_1_Close | 531 | Bit that tells that the first gripper has been closed |
| Open_Gripper_2_Command | 532 | |
| Close_Gripper_2_Command | 533 | |
| Gripper_2_Open | 534 | |
| Gripper_2_Close | 535 | |
| Out_from_Dowel_Pin_Press | 536 | Bit that tells that the robot is away from the press, so that the press can operate in safety conditions |
| Out_from_Runout_1 | 538 | |
| Out_from_Runout_2 | 539 | |
| Out_from_Index_Pallet | 542 | |
| On_Collision | 552 | Bit that tells that the robot has had a collision |
| Return_Mission_Code | DInt starting from 560 | Double integer data sent to the PLC to inform it that a certain task has been performed |
| Mission_Status | DInt starting from 576 | Double Integer sent to inform about the status of the mission code associated to a specific task the robot has to perform |
| Return_Product_Code | DInt starting from 592 | Data Integer sent to return the product code associated to the piece to store all the informations about it |

Figure 2.7: Robot - Output Addresses.

2.4 Other Devices

In addition to the devices listed above, different others instruments have been used, that differ from the others for either the fieldbus or their configuration.

In most of the cases they have been configured from their specific programming tool or directly from the display of the instrument, so that it is only necessary to specify which program execute and start the instrument through simple istructions implemented and managed by the PLC.

2.4.1 Press

The first operation to perform in the unit provides a press to insert a dowel pin in the exhaust cam phaser. To perform this operation a **Kistler Press** has been choosen.

This particular device is controlled by a drive Indramat from Bosch, similar to the ones used to control the motors, but the controller is itself managed by a **Kistler maXYmos**. Thus, in the development environment the required GSD is the one interfacing directly with the PLC, i.e. the maXYmos, that has been also used to program what the press should do.

For the maXYmos 220 Input/Output Bytes are required.

2.4.2 Measurement Instruments

- **OP150**

The first instrument is a software from **TQMitaca**, necessary to perform the runout measurement and no configuration is required to interface the instrument with the PLC.

Unlike what told until now, this one communicates in another way with the PLC.

The software running on a dedicated PC has been programmed with all the informations required to perform the test (mainly mechanical informations and tolerances), and it is the same PC that exchanges informations, by reading and writing in certain areas of memory of the PLC, called DB (Data Block) and briefly explained later on. [1.4](#)

In these areas the instrument writes whether the test is ok or not, while reads determinated informations, such as in which one of the two runout stations perform the test, the start and so on.

- **OP160**

In order for the last unit to perform different tests, two types of devices are needed. They both are **ATEQ** instruments, but each of them has different functionalities during the leak test phase.

The first instrument is an **ATEQ D520** and its role is to maintain a certain pressure and flow during the EOL test. There are two of them for cycle time reasons.

The second one is an **ATEQ F620**, that is a leak tester and that takes care of measuring the differential pressure decay to verify the eventual leaks. In this case, always for cycle time reasons, there are four devices of this kind instead. Despite they are two different devices, both of them have been configured to communicate with 32 Bytes Input and 32 Bytes Output.

2.4.3 Laser and Dataman

The laser is one of the conclusive device of the unit, whose role is to print a QR code in a certain portion of the pieces (depending on the type a different portion is required).

The **Datalogic Laser** used for this task communicates via *Ethernet/IP*.

This means that no configuration inside the TIA Portal development environment has been done. it was only necessary to assign an IP address to the laser through its configuration tool.

A **COGNEX Dataman** is present after the laser station, to verify that the QR code impressed by the laser is actually readable and matches the string passed to the laser to generate it, and to perform a quality measurement of the QR code itself. The communication in this case is still via PROFINET.

Chapter 3

Operation 150

This chapter is dedicated to the explanation of the various stations included inside the OP150 mentioned in the introductive chapter, that depends on the piece tipology the assembly line is processing.

The assembly line can process one tipology at a time, that must be chosen before starting. So, the stations inside OP150 and described later on in this chapter are mutually excluded.

If an *Exhaust* while the *Intake* piece must undergo a Runout test, to check the concentricity of the piece itself.

The next subsection explains the procedure implemented to insert the dowel pin, that is a cilinder shaped piece that is used to connect the rotor of the exhaust cam phaser directly to the camshaft, with a previous description of how the signals and command are managed to create the cycle.

Thus, what is described for the Dowel Pin Insertion is valid for each following station.

3.1 Dowel Pin Insertion

In the *Hardware Configuration* chapter it was briefly described the CPX-CP decentralized system used to connect and manage the pneumatic part of the unit. In the

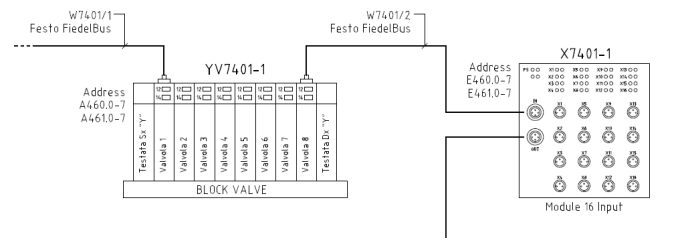


Figure 3.1: CPX-CP System - Zoom on Dowel Pin Block Valve and Module Input.

previous picture (Fig.3.1) are present the block valve to command actuators, with the respective command signals ($A460.0-7$ / $A461.0-7$), and the 16 Input Module to receive feedbacks ($E460.0-7$ / $E461.0-7$) from the sensors connected to it.

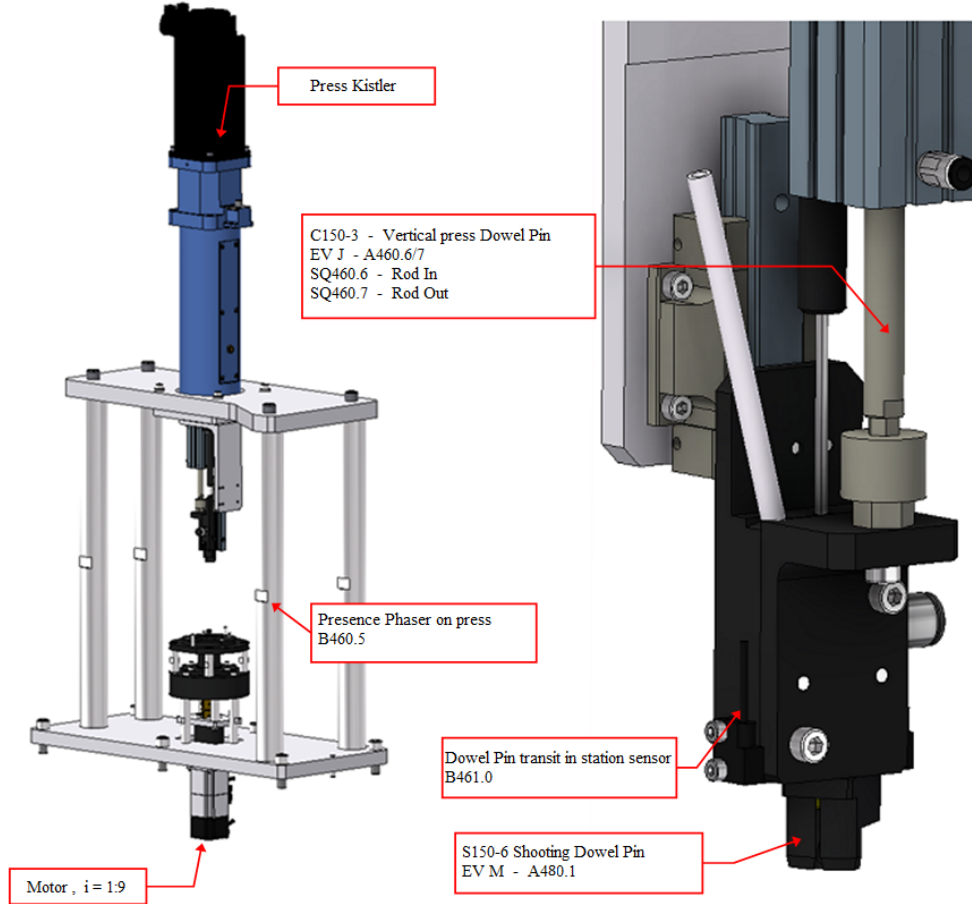


Figure 3.2: Kistler Press - Dowel Pin Insertion.

To the block valve are directly connected solenoid valves, either bistable or monostable, while to the 16 Input Module are connected proximity sensors and photocells, elements that are showed in Fig. 3.2 and that are meant to make the station works. They have been named using particular symbols, to distinguish what they are:

- *EV J* - Bistable solenoid valve

- *EV M* - Monostable solenoid valve
- *SQ* - Proximity sensor (or Reed switch)
- *B* - Photocell

To each of them has been assigned an address, in order to command them or receive their feedbacks.

- **A460.6/7** - Command given to the bistable solenoid valve used to command the vertical cylinder *C150-3*. *A460.6* is to command the cylinder to rod out position and *A460.7* to command it to rod in
- **E460.6** - Proximity sensor signal sent when the cylinder reaches the rod in position
- **E460.7** - Proximity sensor signal sent when the cylinder reaches the rod out position (*E460.7*)
- **E460.5** - Photocell signal used to know if the piece is under the station (*E460.5*)
- **E461.0** - Photocell signal used to know if the dowel pin is in position under the press to be inserted (*E461.0*)
- **A480.1** - Command given to a monostable solenoid valve to blow the dowel pin in position under the press

These input and output signals are not used directly, for different reasons. For example, let's consider an input signal coming from a proximity sensor checking the rod in position of the cylinder. To be really sure that the cylinder is in position, a certain delay is added to the signal after this last arrives (*Delay-On*), so that during the cycle it is not used the original signal anymore, but the delayed one. Another example can be the input signal coming from the pressure switches. Pressure switches have pressure fluctuations, so to have a stable input signal, two delays are necessary.

The first one is activated when the signal arrives, so it is a *Delay-On*, while the

second when the signals goes off, that is a *Delay-Off*. Thanks to these two delays, even if oscillations occur, the signal is at 1.

A similar thing happens for the output signals. Also in this case the reasons can be multiple. Let's try to explain them briefly with the following *KOP* segment showed in Fig. 3.3.

The first reason is that the same output signal, in this case *A460.7*, responsible for the vertical press cilinder to go downward, is used in different situations: manual mode, automatic mode and reset.

The first istruction the PLC runs after the unit is ready to work is checking the operative mode of the unit, selected by an manual selector present on the apposite pushbutton.

There are two *Int* variables: the first one represents the number of station, in this case 150, while the second the manual code, associated to the number of the cilinder, that is 3 for the vertical one. These *Int* variables are used to select the element to be moved directly from a **HMI** (Human Machine Interface) panel present on each unit. The first variable is used to tell which station to control, while the second which element of the station to move.

If these two conditions verify, then the PLC checks another manual selector, used to perform the movement. This selector is used to choose where to move the cylinder (in case of a motor it's utilized to make the motor move in Jog+ or Jog-): if it's set to *Minus* the cylinder moves to rod in, viceversa if it's set to *Plus* the cylinder moves to rod out.

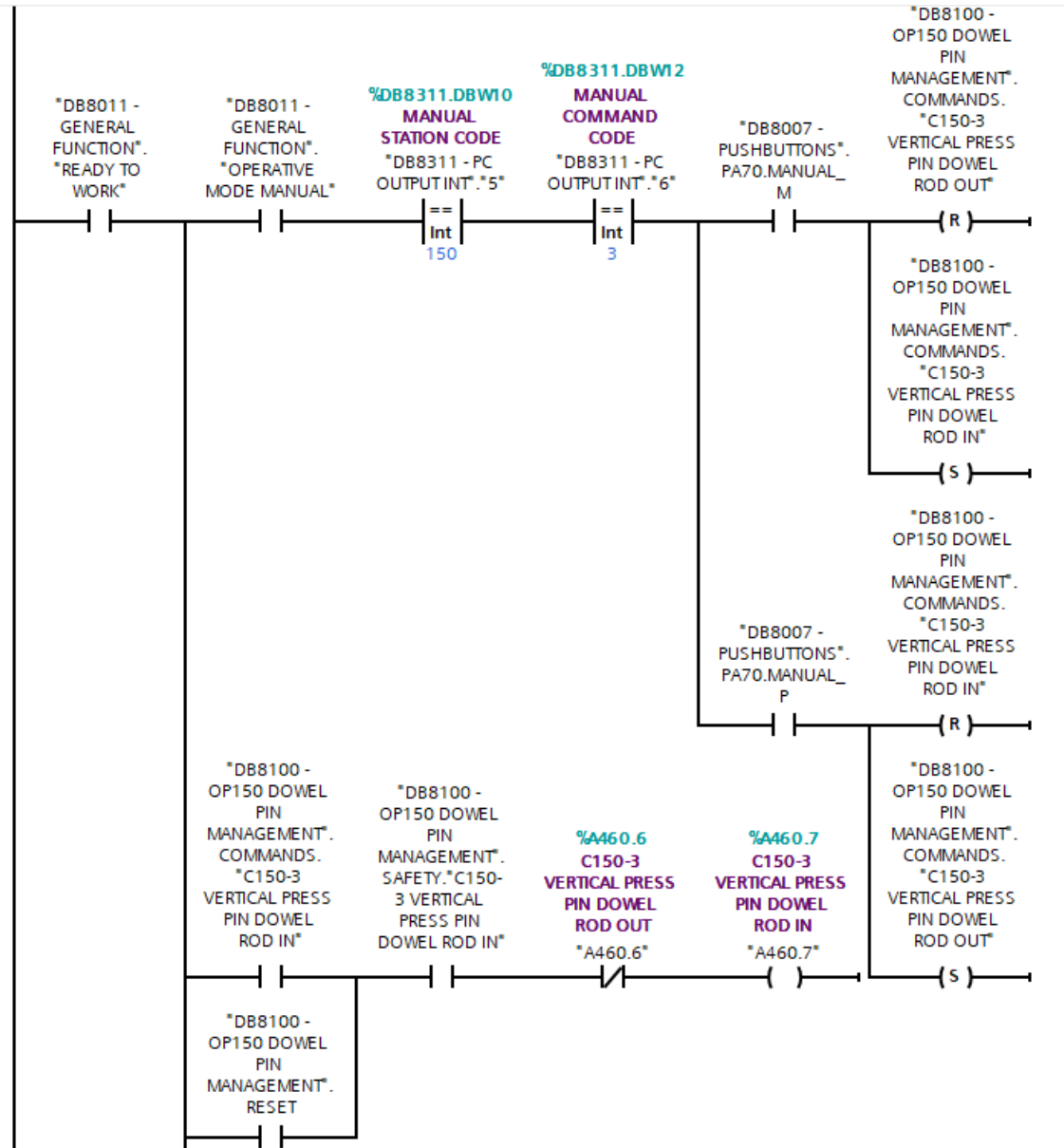


Figure 3.3: Indirect Command Signals

Considering the *Minus* selector is selected, a boolean variable corresponding to the rod in command is set, while another boolean variable corresponding to the rod out command is reset.

If the boolean variable is set and there is the safety for the execution of the movement, the effective output command is given, as showed with the last line of the segment.

This KOP segment is used also for giving the direct command (*A460.7*) during the automatic cycle, or in case of a *Reset*. These two situations happen during the normal cycle of the unit and are described in the following subchapter.

The ‘*Indirect*’ commands, together with other signals, such as the delayed ones coming from the proximity sensors or particular safety conditions signals, are stored inside a particular DB, called **Management DB**. There is one management DB for each station cycle that has been implemented.

Safety condition signals are mainly mechanical conditions, so particular variables have been created to avoid mechanical parts to collide implementing KOP segments to interlock, for instance, particular actuators movement, or to wait until a robot is out of space.

Other safety conditions come instead from particular photocells (like the *B460.5* showed in Fig. 3.3, that tells if a particular movement can be performed or not. If a piece is under a station to be processed, the robot cannot place another piece under the same station. Before starting the station cycle, then, it is firstly necessary to create all these variables inside the Management DB, manage them, and in the end use them to create the cycle of the station they are part of.

3.1.1 Cycle Description

The *Exhaust* cam phaser has a bias spring that permits more rapid respond of the rotor or helps the camshaft to be placed in particular orientations during the engine’s working cycle.

To perform the station cycle a *Int* variable is created. This variable is used as a register, so that each time an operation is completed this register’s value is increased

and the next step of the cycle can be performed.

Every time a new step is performed the register value is checked to control that the step can be effectively performed. This is also a way to understand eventual problems that can happen during the normal cycle of the station. If the register value does not match, the cycle cannot continue, so there is a problem earlier on and it is more easy to understand what the origin of the problem.

Each step corresponds to a specific segment implemented inside a function, called inside the *Main Program Cycle* to be performed cyclically.

This register value is obviously set to 0 in order for the first step to be performed and it is incremented by 10 at each step.

- *Step 0* - This step corresponds to the *Start Cycle*.

Moreover, in order for the cycle to start, the station must be in '*Home Position*', that is a particular rest condition that needs to be verified. This condition is important to be sure that the previous cycle has been completed and it is also a particular condition for when a reset is necessary.

When a *Reset* button is held for more than 3 seconds, the station automatically moves to the Home Position, to start again with the cycle once a *Start* button is pressed.

This Home Position corresponds to the cylinder *C150-3* in rod-in and the motor in a particular known position. If this happens the register value is incremented and the previous results of the dowel pin insertion, given by the press controller and that tells the PLC whether the dowel pin has been inserted correctly or not, are reset.

This reset is not necessary, but is done to be sure that errors due to incorrect data reading are not present.

- *Step 10* - This step checks that the robot is actually out of space.

The robot cycle is not included into the station's cycle, nor into the one of other stations. Each robot has its own cycle run by its controller and programmed inside its programming tool.

Through specific signals sent to the outside and specified earlier (Fig. 2.7), the PLC knows exactly if the robot has performed a specific operation.

If the *Out.From.Dowel.Pin.Press* signal is equal to 1, the robot is out of space,

so the station has the permission to continue with its cycle.

The motor can now move to load the spring and the next step can be performed.

- *Step 20* - The motor loads the bias spring moving to a specific position.
- *Step 30* - The cylinder C150-3 can move to rod out position. This cylinder is used to put the press in the right position to insert the dowel pin.
- *Step 40* - A function is called to send the program to be performed by the press. This program number corresponds to the program effectively implemented through the software tool of the *Kistler MaXYmos* controller.
- *Step 50* - A start command is set for the *MaXYmos Controller* to execute the program. The press goes to a pre-insertion position.
- *Step 60* - This step verifies that the motor has effectively loaded the spring and that the press is in the right position. If this happens the cycle of the press is paused by resetting its start command.
- *Step 70* - The presence of the dowel pin is checked
- *Step 80* - The press start command is set again, together with a ‘*Continue*’ command, so that the press inserts the dowel pin.
- *Step 90* - The start and continue commands of the press are reset.
- *Step 100* - The cylinder *C150-3* move back to rod in position.
- *Step 110* - The motor moves back just enough to make sure that the piece can be properly removed
- *Step 120* - The PLC gives to the robot the consent to pick the piece from the station and the mission code to be performed.
- *Step 130* - The robot must pick the piece and go out of space. If these two conditions happen the next step is done.
- *Step 140* - The motor moves back to the starting cycle position, so the cycle can start again.

3.1.2 Feeder Management

The dowel pin is inserted in a specified position by a press, but before must be positionated under this last one.

To do that a circular vibrator has been utilized, that has its own cycle to position the dowel pin correctly and that's interlock with the insertion cycle previously described. Also in this case actuators and valves are present (Fig. 3.4).

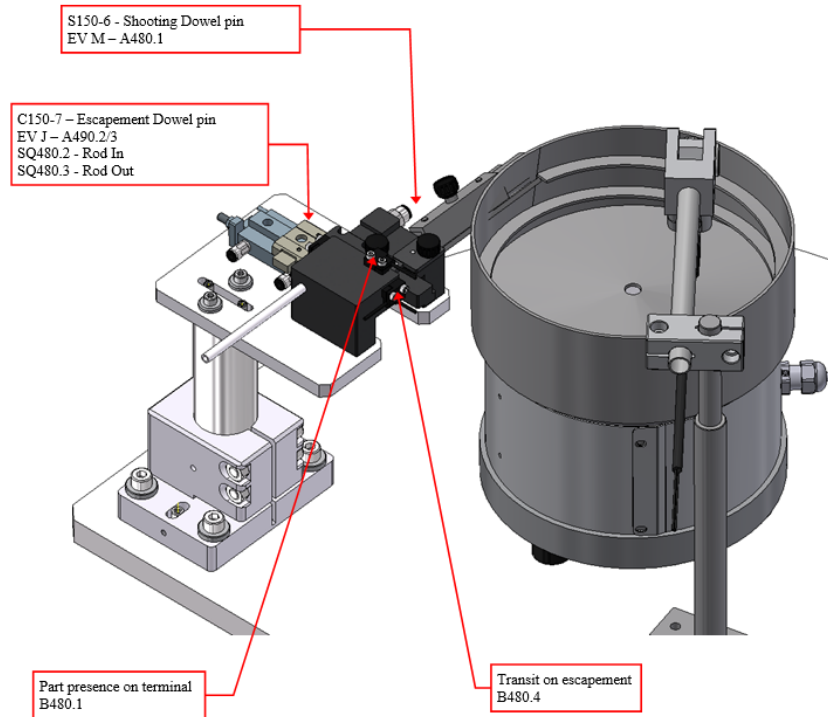


Figure 3.4: Dowel Pin Feeder.

The same approach of the previous cycle has been followed, so an integer value has been created to perform the various steps.

The dowel pin is positionated under the press by an air blow, that shoots the element inside a white tube that arrives directly under the press (see Fig. 3.2).

The first thing to do is having the vibrator in starting work position, that means having the *C150-7* cylinder, responsible for moving the dowel pin to be correctly shooted inside the tube, in rod in position.

The vibrator constantly vibrates to make the dowel pin reach the position in which

must be moved by the cylinder to be positioned in front of the air blow. If it's on correct position, a photocell (*B480.4*) signal goes from 0 to 1.

Before the cycle can continue, there must not be another dowel pin under the press, and that is checked through a specific photocell, that is *B460.1*.

If no other element is present under the press, then the *C150-7* cylinder can escape the dowel pin, by commanding the bistable valve with the command signal *A490.2* that moves the cylinder to rod out position.

At this point, by commanding the monostable valve through its command signal (*A480.1*), the dowel pin is blown into the tube, to reach its position under the press.

Since this is the last station that actually insert a component, there is only one feeder in the unit. However, there are many of them distributed along the assembly line, each of which is responsible for getting pieces in the correct position to be subsequently inserted as specified.

3.2 Runout Test

This particular test must be performed if an **Intake** cam phaser is under production.

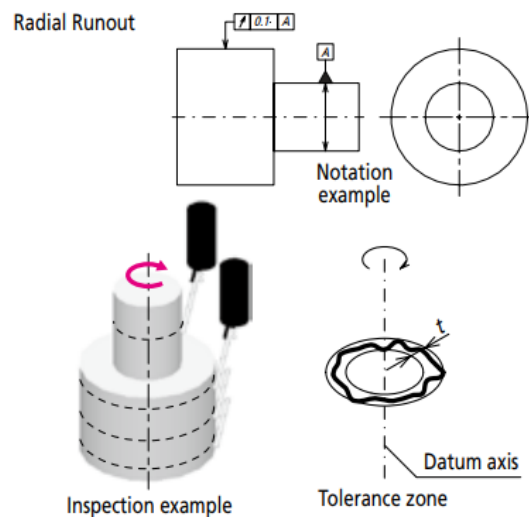


Figure 3.5: Circular Runout. [10]

Making reference to Fig.3.5, the test is done to calculate the **circular runout**,

which aims to check if the line is contained within the tolerance zone formed between two coplanar and/or concentric circles perpendicular to a datum axis.

The *TQM* bench performs a dynamic measure: the measuring system manages the rotation of the parts on a precision mandrel, controlled by an encoder.

The system uses only 1 probe for each section and acquires at least 3.600 couple of point for each section.

With these points the TQM software re-creates the real profile of each sections and it perform the measures.

To perform the test the instrument requires a reference, that can be either a section or a plane. The instrument automatically relates the measures of each section with the reference.

Moreover, on the reconstructed profiles it is possible to do the Fourier analysis and it is also possible to measure particular characteristics created with more sections, such as concentricity, cylindricity, flatness, parallelism and so on.

Concentricity is particularly important, since it is the only parameter required from the customer to distinguish a good piece from a reject one. The concentricity

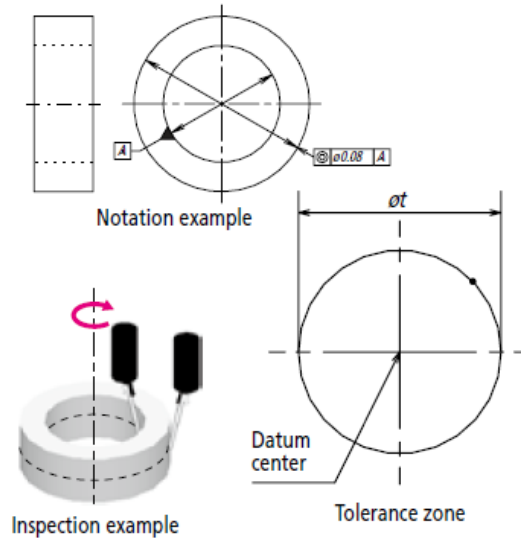


Figure 3.6: Concentricity. [10]

(Fig. 3.6) is different from the circular runout, since aims to understand if the center

point is contained within the tolerance zone formed by a circle of diameter concentric with a certain datum, in this case an axis.

However, the runout can be seen as a combination of concentricity and circularity.

$$\text{Runout} = \text{Circularity} + \text{Concentricity} \quad (3.1)$$

If the part is perfectly round (that means perfectly circular), the runout equals the concentricity.

In order to better understand how the procedure works, let's have a look at the following picture (Fig.3.7). The cyan colored part is the *outer plate*, while the pur-

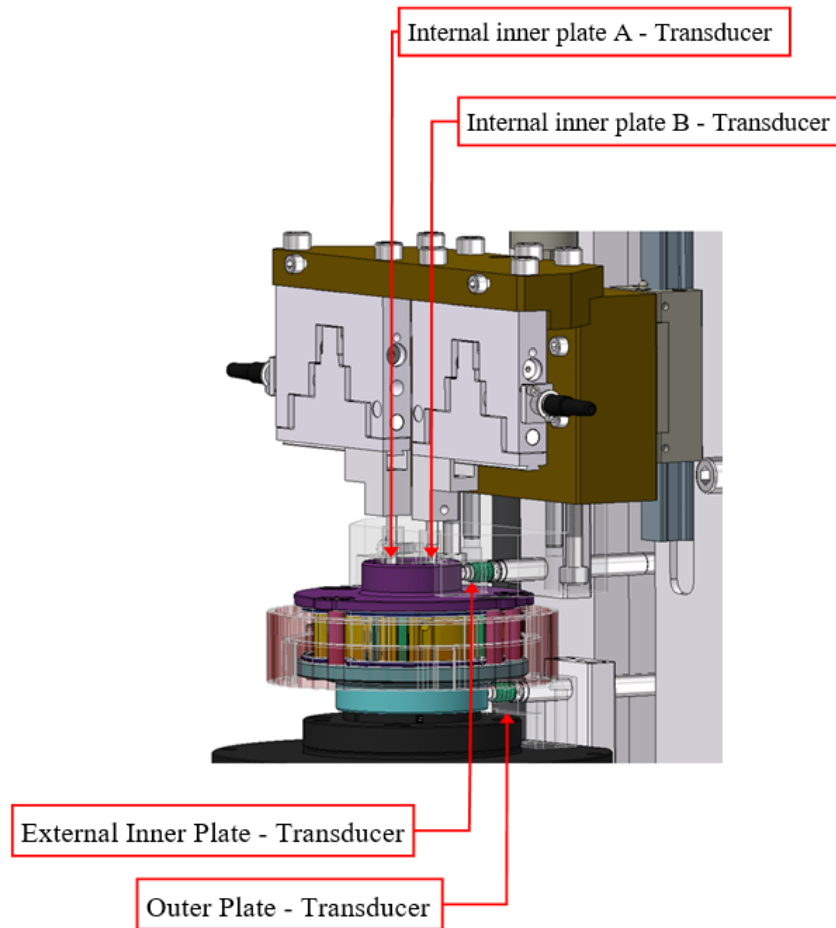


Figure 3.7: Runout Station - Transducers.

ple one is the *inner plate*.

These two parts, together with the sprocket, are fastened together by a previous operation and together form the stator.

Let's call the internal diameter of the inner plate **A** and the external diameter of the outer plate **B**.

To perform the test the parts must be blocked and centered under the instrument by a pneumatic system with 3 pliers that work on the internal diameter of "B".

In the external diameter of "B" there is a probe (or transducer, like showed in the picture) that measure the section of the diameter itself.

The other 3 probes are: one to measure the external diameter and the other two for the internal diameter of "A".

The software creates the axis of "A" and measures the Runout of the two external diameters with respect to "A" (indicatively the runout of the external diameter of **B** must be lower than 0.2).

To mark the piece as good the concentricity to be checked is the one of the external diameter of "B" with respect to "A". If the concentricity is less than 0.6 than the piece is considered good.

The instrument requires all the mechanical specifications to perform an adequate measurement, so the diameters of the surfaces the transducers are directly connected to, and the tolerances to satisfy, such as the required concentricity, angular tolerances and all the other tolerances specified from design, so that it automatically marks the piece as reject or not.

In addition to that, it is necessary to exchange informations with the instrument.

As anticipated in chapter 2 this happens through a dedicated computer that runs the program responsible of managing the instrument.

This PC exchange informations with the PLC in a transparent manner, by reading and writing in a specific *DB*.

This happens thanks to a library called **LibNoDave** that is responsible of this data exchange.

The DB looks like the following picture (Fig. 3.8). The elements inside the data block are used to send commands or receive signals, as explained for all the other components in chapter 3.

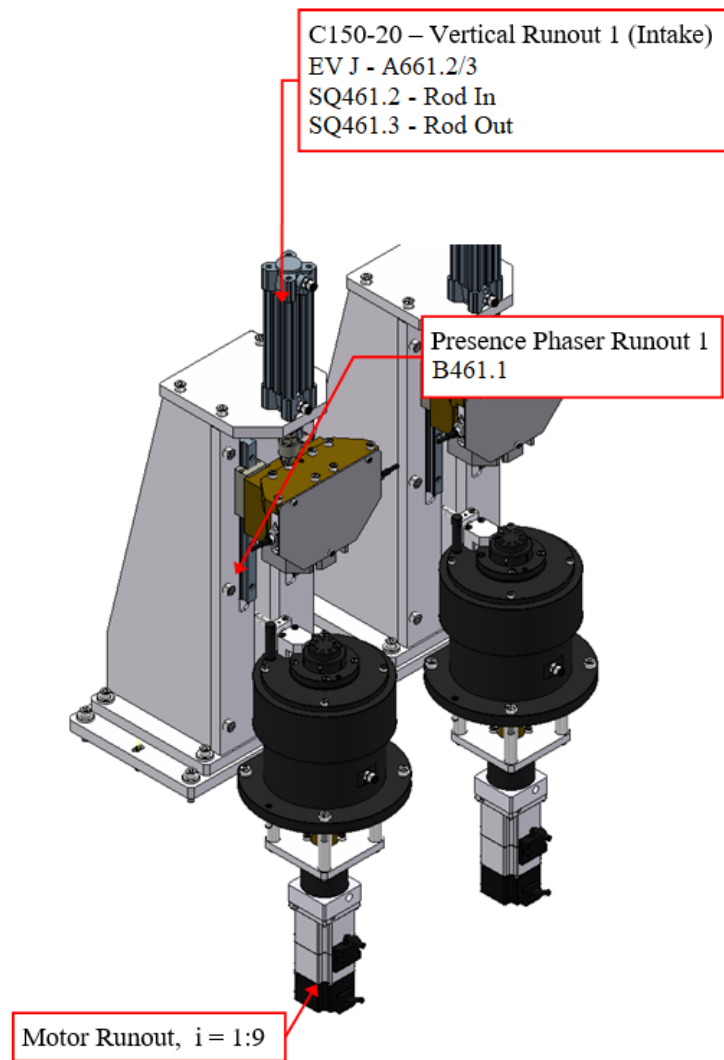


Figure 3.9: Runout Station.

The cylinder *C150-20* can move downwards and the pliers block and center the piece, then the two upper probes can move to rod in, so that they go directly in contact with the internal diameter of the inner plate. The other two probes, instead, are directly in contact when the robot places the piece under the station.

The motor can now start moving so that the probes can acquire all the 3.600 couple of points needed. To do that the motor must do a little more than 360° to assure a correct measurement.

Once the instrument finishes it automatically toggles a specific bit inside the data block to inform the PLC if the piece is good (so the *RESULT MEASURE - OK* signal passes from 0 to 1) or not (viceversa the *RESULT MEASURE - KO* passes from 0 to 1), that is crucial to understand if the piece must undergo other tests or must be rejected from the line, while the result data are directly send to the customer SQL server and are not stored inside the PLC.

To complete the cycle the motor moves back to the start cycle position, the inner plate internal probes are commanded to rod out, the pliers are removed and finally the cylinder *C150-20* is moved back to rod in position.

At the end of the test, the instrument plots automatically a runout graph.

In Fig. 3.10 it is possible to see the runout of the external diameter of **B**.

Above in the picture, the runout is represented in polar coordinates, while, below, in cartesian coordinates.

The first one is fairly descriptive, but can be mixed with the geometrical shape of the piece. The second one, instead, shows the displacement versus the angular position. The runout measure is given by the difference between the maximum value of the displacement and the minimum one, as showed with the red line in the picture.

In this case the runout seems to be approximately equal to 0,0165.

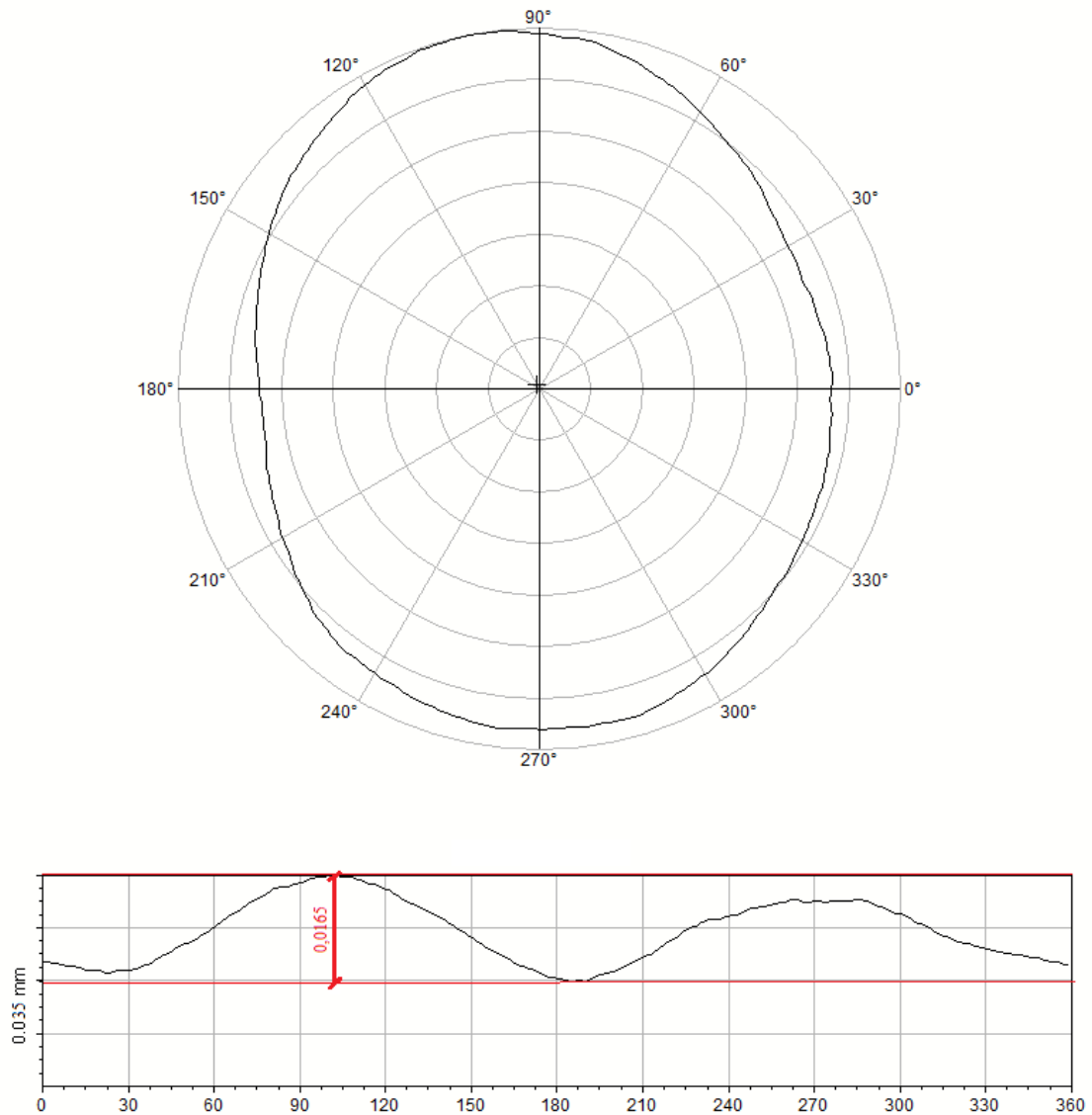


Figure 3.10: Runout in polar (above) and cartesian coordinates (below).

For what said before, however, the **concentricity** is the parameter to be considered to mark the piece as good.

These two parameters are related, so, typically, if one is not respected, neither is the other one (see [3.1](#)).

Chapter 4

Operations 160 and 170

In the previous chapter it was described the first test executed on the assembled pieces, test that is performed, however, only on a specific typology, that is the *In-take* one.

In the first part of this chapter are described the two other tests, present in the operation 160 and that are applied to both the assembled typologies.

The first test performed is a **EOL** test, that is meant to determine if there are defective pieces or specifications are not matched by simulating critical working situations. There are two *EOL* stations.

The second one is a **Leak** test. In this case the test aims at detecting manufacturing defects that correspond to a leakage of gas from the product.

Indeed, although the preproducts are hydraulic phasers, the leak test is performed using air. This has been done mainly for costs reason, but also because it is particularly difficult to maintain stationary conditions to perform the test using oil, especially for what concerns the temperature. Also in this case there are more stations, in particular 4 of them.

The second part of this chapter is dedicated, instead, to the last operation to be performed, that is operation 170.

This one is responsible of marking with a laser the pieces declared good by the previous stations.

4.1 Operation 160

4.1.1 EOL Test

The *End-of-Line (EOL)* test is a specific and critical step for an assembly line, because it assesses not only the quality of the product, but also the yield of the production process.

The defective products that do not match the specifications closely enough must be marked as rejects and separated from the good ones.

Moreover the results of the testing should be repeatable under the same conditions and reproducible in any other location using the same instrument, but more importantly stable over time and free of bias [11].

In order to do so a specific test cycle has been designed and an *ATEQ* instrument has been used to simulate working conditions. The **ATEQ D520** is not used only to perform measurements (flow measurements are performed in working conditions), but also as a sort of proportional valve to set two different pressures during the cycle test.

Also in this case it was necessary to design different programs, directly from the display of the instrument, that are then passed to the instrument itself during the cycle and started when needed.

So a simple function block has been implemented to exchange these data.

The function is exactly equal to any other function that deals with the management of the instruments: the first part is responsible of transferring and storing informations (from or inside the data block), while the second part regards its general behavior.

In this case it was necessary to create an instrument cycle, where it is possible to select the program (created directly from the monitor of the instrument), to start and stop it, so that the EOL test can be performed according to the specifications.

The EOL test aims, as told before, at testing the quality of the product by simulating working conditions and checking that everything works correctly.

As briefly explained in the previous chapters, the cam phasers are composed of two main parts: a stator and a rotor.

A vaned rotor is disposed inside a lobed stator and they are mechanically locked together by a **locking pin**. In this position, which is the default position, a spring biases a cylindrical locking pin outward to engage a pin bore disposed in the stator. In this condition the stator and the rotor are coupled and cannot move one with respect to the other.

So if I want the rotor to move with respect to the stator, the locking pin has to be unlocked and this happens when the oil pump pressure reaches a pre-determined level, so that the hydraulic force of the oil, through a suitable porting, causes the locking pin to retract from the pin bore and to decouple the rotor from the stator. Thus, depending on the conditions, the locking pin needs to be unlocked or not. For example under conditions of low engine oil pump pressure (ad example during the startup), it is desirable to mechanically lock the rotor and stator together (in default mode) to prevent unwanted relative angular movement of the rotor/stator when the pump pressure is not high enough to reliably position the rotor relative to the stator.

To better understand how the overall procedure has been implemented, let's have a look at the EOL station (Fig. 4.1).

The piece is placed under the station, that must be equipped according to the type of piece, since they have different shapes (there are several sensors to ensure that the station has been properly equipped to avoid damages).

As visible in the picture, there is also a sensor that checks if the piece is under the station (*Presence Phaser EOL 1*). This is to avoid, instead, to put another piece under the station when there is another already, or to perform a test when no piece has been placed.

After the piece has been placed it is necessary to block it, so that a motor is able to move the rotor simulating particular situations, and this is done by two horizontal slides. At this point it is possible to move the vertical cylinder *C160-70* so that the cycle can begin.

To perform the tests, unlike what happens in practice, **air** was used instead of oil.

Two different air entrances are necessary. The first one is directly above the piece,

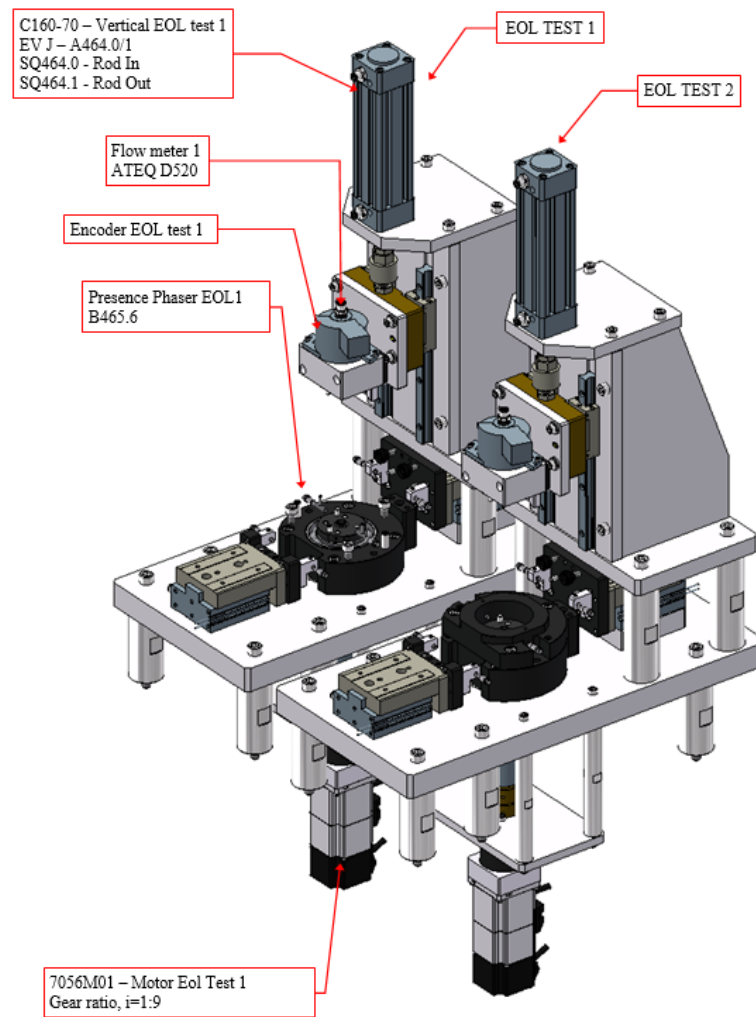


Figure 4.1: EOL Test Stations.

the position of the rotor during its movement, and a torque meter to calculate the torque exchanged between rotor and stator in particular conditions.

4.1.1.1 Test cycle

The applied test cycle is essentially the same for both tipologies. The only difference is in the direction of rotation of the rotor depending on the fact that the stator housing has a different shape in the two cases.

To give an idea of the results, these tests have been performed on 10 different intake pieces (unfortunately the exhaust ones are not showed since a mechanical retooling was necessary), and, in the following lines, are reported the average values of the results.

The tests applied, in order of application, are:

- **1: Locking pin locking control:**

The first condition to verify is if the locking pin is unlocked only when a certain pressure is reached.

The locking pin should unlock when a pressure approximately equal to $0.5 - 0.6$ bar is applied. Thus, if a lower pressure is applied, the unlocking should not happen.

And this is what has been done: a first program on the Ateq has been created to set the pressure to 0.3 bar and the second solenoid valve is commanded in order to have the flow coming from "A" (from under the piece).

This first test is not over yet, because the rotor is then moved in both direction, untill mechanical stop, with the motor set to move with a fixed torque equal to 2 Nm to check that the mutual movement between the stator and rotor is effectively blocked.

Each time the rotor reaches the mechanical stop, the position measured by the encoder is saved, so that is possible, with a simple difference, to calculate the free angle of the locking pin, that must be contained in a certain window (approximately 1°).

- **2: Locking pin unlocking:**

The program number is now changed to vary the pressure at *0.6 bar* so that the locking pin can be unlocked.

The same procedure described for the previous test has been implemented, but this time the rotor is free to move with respect to the stator, so that the wide rotor vane can actually touche the stator lobes (the other vanes and lobes have extra clearance to prevent contact regardless of rotor position).

The wide angle vane is better able to sustain the shock of impact when a vane strikes a lobe in an uncontrolled event such as at engine start-up.

When the wide rotor vane touches both the contact surfaces of the stator, the position measured by the encoder is saved, so that it is possible to calculate the total rotor displacement angle (normally within *20-30°*).

This is done saving the two contact positions measured with the encoder and calculating the difference.

The encoder measurement showed that the assembly was correct, since the average value of the displacement is **25,78°**.

Moreover, during all the relative movement between rotor and stator, the frictional torque is acquired.

$$0,43 \text{ Nm} < \text{frictional torque} < 0,57 \text{ Nm} \quad (4.1)$$

The average torque measured is **0,54 Nm**.

- **3: Flows control:**

Another program has been implemented to set a specific pressure (approximately *1 bar*) to perform flow measurements.

The Ateq D520 is a flow meter which measures a drop in pressure with a differential sensor (transducer) which is placed at the extremities of a calibrated flow tube.

When fluid (gas) moves through a calibrated flow tube (laminar flow), a drop in pressure occurs, the value of which is proportional to flow.

$$\Delta P = 8\mu LQ/\pi R^4 \quad (4.2)$$

μ : viscosity of the fluid

L : length of the calibrated flow tube

R : radius of the calibrated flow tube

Q : **flow**

ΔP : drop in pressure in the calibrated flow tube

So it is possible to obtain directly the flow Q by the previous formula.

The first flow measurement performed is done commanding the second valve (from left in Fig. 4.2) and the third (from left), so that the air flows directly from "B", which is above the piece, and is directly exhausted from "C".

The rotor is moved in different position during this phase of the test, to consider different working conditions. Each time the limit values change.

The first time the rotor is moved approximately of 2° .

$$36 \text{ lmin} < \text{Flow } B < 40,5 \text{ lmin} \quad (4.3)$$

The average *Flow B* is **38,344 lmin**.

Then the rotor is moved in middle position, which is half the displacement angle (that is about 25°).

$$37 \text{ lmin} < \text{Flow } B < 43,7 \text{ lmin} \quad (4.4)$$

The average *Flow B* in middle position is **40,4 lmin**.

Now the flow is changed, by releasing the command of the second valve, and commanding the first one. In this case the air flows from "A" to "C" and the same procedure explained previously is applied.

To perform this flow measurement the rotor is moved almost the whole displacement angle, and this time:

$$36 \text{ lmin} < \text{Flow } A < 50 \text{ lmin} \quad (4.5)$$

The average *Flow A* is **43,373 lmin**.

At the end the rotor is moved to the rest position.

- **4: Locking pin locking:**

The last part of the cycle consists of placing the locking pin in the default position.

The air flows and pressure are then removed and it is verified, again, that the rotor is effectively blocked.

4.1.2 Leak Test

Leak Test is the last test performed on the products.

In the introduction of this chapter has been anticipated that this test is performed with air, even if the test piece is actually hydraulic actuated.

This is possible due to the physical properties of liquids and gases. The higher the viscosity of the liquid, the slower it will leak through a particular hole.

Since the viscosity of a gas is generally much lower than that of a liquid (the viscosity of air is about 50 times lower than that of water), if we consider the influence of the viscosity only, a particular leak that passes 20 *mm*³ of air per second, would only pass 0.4 *mm*³ of water per second.

Moreover the oil has viscosity that is even higher than that of the water, that leads to a even bigger reduction of leak with respect to that of the air.

Furthermore the viscosity is very sensitive to temperature variations. As showed in the previous table (Tab. 4.1) the higher is the viscosity the higher is the sensitivity to the temperature variations and the oil undergoes a sharp change if its temperature varies between 20 and 70°C.

In order for the test to be correctly performed, thus, the oil must be kept at a fixed temperature and this is very expensive in terms of costs.

The air, instead, does not change its viscosity in a significant way and it is easier to maintain its temperature constant.

These are the main reasons for which the air has been preferred to the oil.

| Fluid | Temperature | Viscosity |
|-------|-------------|---|
| Air | 0°C | $1.71 \times 10^{-5} \text{ Pa s}^{-1}$ |
| | 20°C | $1.81 \times 10^{-5} \text{ Pa s}^{-1}$ |
| | 50°C | $1.95 \times 10^{-5} \text{ Pa s}^{-1}$ |
| | 70°C | $2.04 \times 10^{-5} \text{ Pa s}^{-1}$ |
| Water | 0°C | $1.79 \times 10^{-3} \text{ Pa s}^{-1}$ |
| | 20°C | $1.0 \times 10^{-3} \text{ Pa s}^{-1}$ |
| | 50°C | $0.55 \times 10^{-3} \text{ Pa s}^{-1}$ |
| | 70°C | $0.40 \times 10^{-3} \text{ Pa s}^{-1}$ |
| Oil | 20°C | $2.6 \times 10^{-2} \text{ Pa s}^{-1}$ |
| | 50°C | $1.0 \times 10^{-2} \text{ Pa s}^{-1}$ |
| | 70°C | $0.7 \times 10^{-2} \text{ Pa s}^{-1}$ |

Table 4.1: Viscosity Coefficient of Gases and Liquids. [13]

This test consists of filling the piece to a defined overpressure with air. After filling the test piece it is always necessary to wait before measuring until the parameters have stabilized and the pressure has settled, that usually takes longer than the actual measurement.

The pressure in the test piece is then measured over a defined time interval: if the pressure reduces over time, there is a leak.

For this purpose an *Ateq F620* has been chosen, which is a compact air/air leak detector.

This instrument can detect a small variation (or drop) in differential pressure between the test and a reference, when both are filled to an identical pressure.

The station used to perform the cycle is represented in the next picture (Fig. 4.3).

There are four stations, each one with its own Ateq instrument.

The piece, moreover, is graphically modified since it is sensitive material.

Once the robot has placed the piece on the horizontal slide (*C160-20*), this last is moved to rod in position to start the test. Then, the vertical one (*C160-21*) is commanded to rod out position, so that the leak measurement can start.

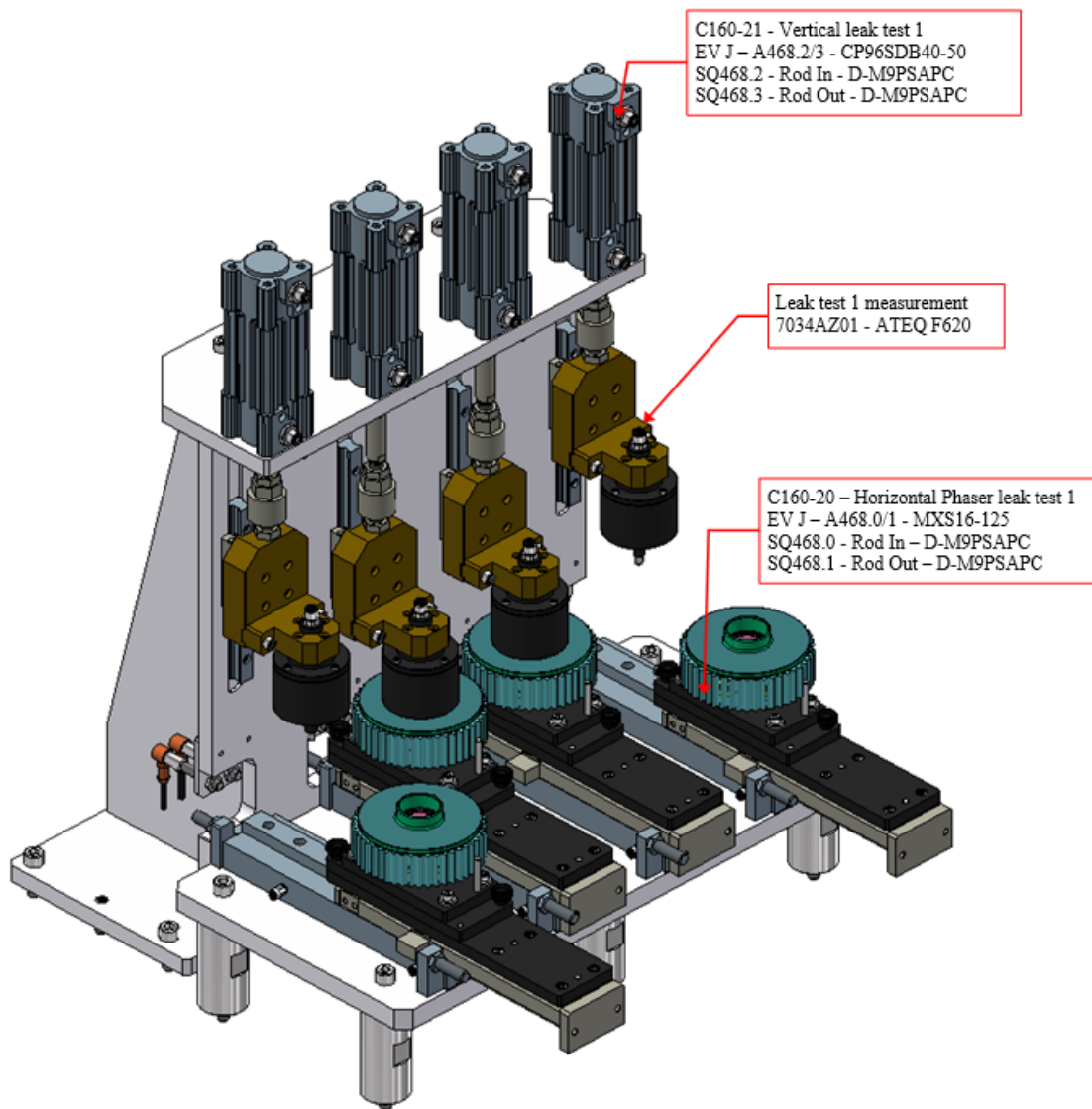


Figure 4.3: Leak Test Stations.

The leak test is performed twice for each piece: the steps are the same, only the pressure applied is different.

First thing the instrument is started so that the piece is filled with air at a certain pressure. This step requires a time called **filling time**, followed by another time, called **stabilization time** instead. Usually these times are the most expensive in terms of durations and are in the order of seconds.

The instrument performs then the test over a defined time interval, called **test time** and the differential pressure is calculated.

The same procedure is performed again with a higher pressure before moving the two slides and removing the piece from the station.

Unfortunately, also in this case, it was not possible to store data, because the final client was still considering the pressure values to perform the test at, and most importantly the leak values to distinguish a good piece from a reject.

Leak values depend mainly on the way in which the seal is inserted.

Indeed, the seal insertion is one of the fews operation to be performed manually, since, because of the shape and the material, it is practically impossible to insert it other ways.

Therefore, positioning errors, no matters how small they are, can greatly vary the leak values.

4.2 Operation 170

The previous test operations are necessary to identify the good pieces and the bad ones: the bad must be removed from the line, while the good ones can be marked with their own serial number and with a data matrix code. In the previous picture (Fig. 4.4) is showed the last station present in the unit.

The station is composed of a rotary table with two positions and of a Datalogic Laser.

To mark the piece it is necessary to place it on the table and rotate it, so that the piece is moved directly in position where it has to be marked.

Moreover, it is necessary to close the safety guard of the laser (the black box showed in the picture) in order for the operation to be performed. This safety guard is

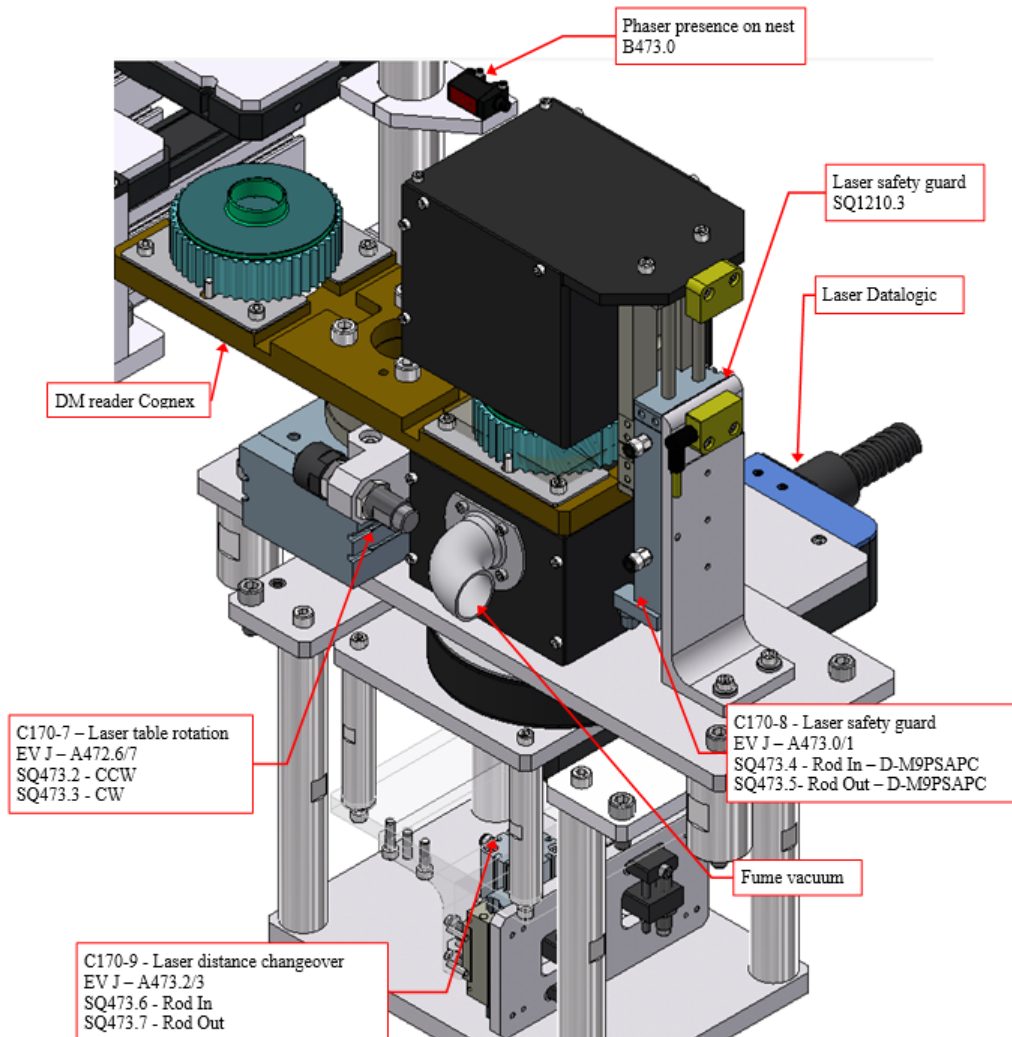


Figure 4.4: Laser Station.

directly commanded by a solenoid valve and there is an apposite safety switch to check whether the safety guard is open or not.

The laser marks a date-time format string (*yyyy/MM/dd/hh/mm/ss*), followed by a serial number, and a data matrix containing the previous string on the lower part of the piece.

Once the laser finishes, the table rotates again, so that a Cognex DataMan can read the data matrix code impressed by the laser and performs a quality measurement.

Depending on the piece tipology, the laser has to mark in different positions, so it has been necessary to create two different programs.

These two program have been programmed into the dedicated PC connected directly to the laser (the PC acts also as a controller for the laser) and consist of two different graphical layouts, saved as *.xlp*.

To select the one corresponding to the correct tipology, the PLC has to send a string to specify which is the correct layout file.

```

1  "DB8953 - OP170 - DB TCP LASER - BUFFER TX".Dato[1] := BYTE_TO_CHAR(16#1B); // Escape
2  "DB8953 - OP170 - DB TCP LASER - BUFFER TX".Dato[2] := BYTE_TO_CHAR(16#0C); // Lenght
3  "DB8953 - OP170 - DB TCP LASER - BUFFER TX".Dato[3] := BYTE_TO_CHAR(16#00); // Lenght
4  "DB8953 - OP170 - DB TCP LASER - BUFFER TX".Dato[4] := BYTE_TO_CHAR(16#F2); // Open file
5  "DB8953 - OP170 - DB TCP LASER - BUFFER TX".Dato[5] := BYTE_TO_CHAR(16#82); // Open file
6  "DB8953 - OP170 - DB TCP LASER - BUFFER TX".Dato[6] := 'I';
7  "DB8953 - OP170 - DB TCP LASER - BUFFER TX".Dato[7] := 'N';
8  "DB8953 - OP170 - DB TCP LASER - BUFFER TX".Dato[8] := 'T';
9  "DB8953 - OP170 - DB TCP LASER - BUFFER TX".Dato[9] := '.';
10 "DB8953 - OP170 - DB TCP LASER - BUFFER TX".Dato[10] := 'x';
11 "DB8953 - OP170 - DB TCP LASER - BUFFER TX".Dato[11] := 'I';
12 "DB8953 - OP170 - DB TCP LASER - BUFFER TX".Dato[12] := 'p';
13 "DB8953 - OP170 - DB TCP LASER - BUFFER TX".Dato[13] := BYTE_TO_CHAR(16#0D); // CR
14 "DB8953 - OP170 - DB TCP LASER - BUFFER TX".Dato[14] := BYTE_TO_CHAR(16#0A); // LF

```

The first 5 lines of the previous code are only sent to the laser to inform it that a new file has to be opened, file that is specified later by creating a new string. In this case the string is *INT.xlp*.

The last 3 lines are, instead, to communicate that the string is finished.

Once the layout has been selected, it is necessary to send, in a similar manner, another string corresponding to what the laser has to mark on the pieces.

Chapter 5

Software Implementations

In this chapter are described the main implementations that allowed to create the operations previously described.

They permit to manage the devices that implement the machine safety, to configure and control motors and robots, to create an exchange of informations about the piece being processed, and finally allow the operator to interact directly with the unit.

5.1 Safety Management

The first thing to do before starting programming is to define all the inputs and outputs connected to the modules of the ET200SP and CPX, listed in the previous chapter, and assign them an address, depending on the module they are connected to.

Let's consider as example the third of the five input fail-safe modules (2.1).

Its starting Address is 1220, so all the inputs connected to this module, depending on the data type, will start from that address untill reaching the last available one. In this case the inputs are booleans and correspond to the ones coming from the safety doors. The correspective addresses are E1220.0, E1220.1, E1220.2, ..., E1220.7, E1220.0, E1220.1, E1221.7, where E is the german mnemonic to indicate an input (A is for the outputs instead). This has been done for all the modules present, whether they are fail-safe or not. Now that the addresses have been prepared, they have to be managed: the Main Safety Function Block manage all the fail-safe inputs and outputs.

To better organize the work several functions have been created, one for each unit for managing each unit's safety, and a single function that is responsible of the

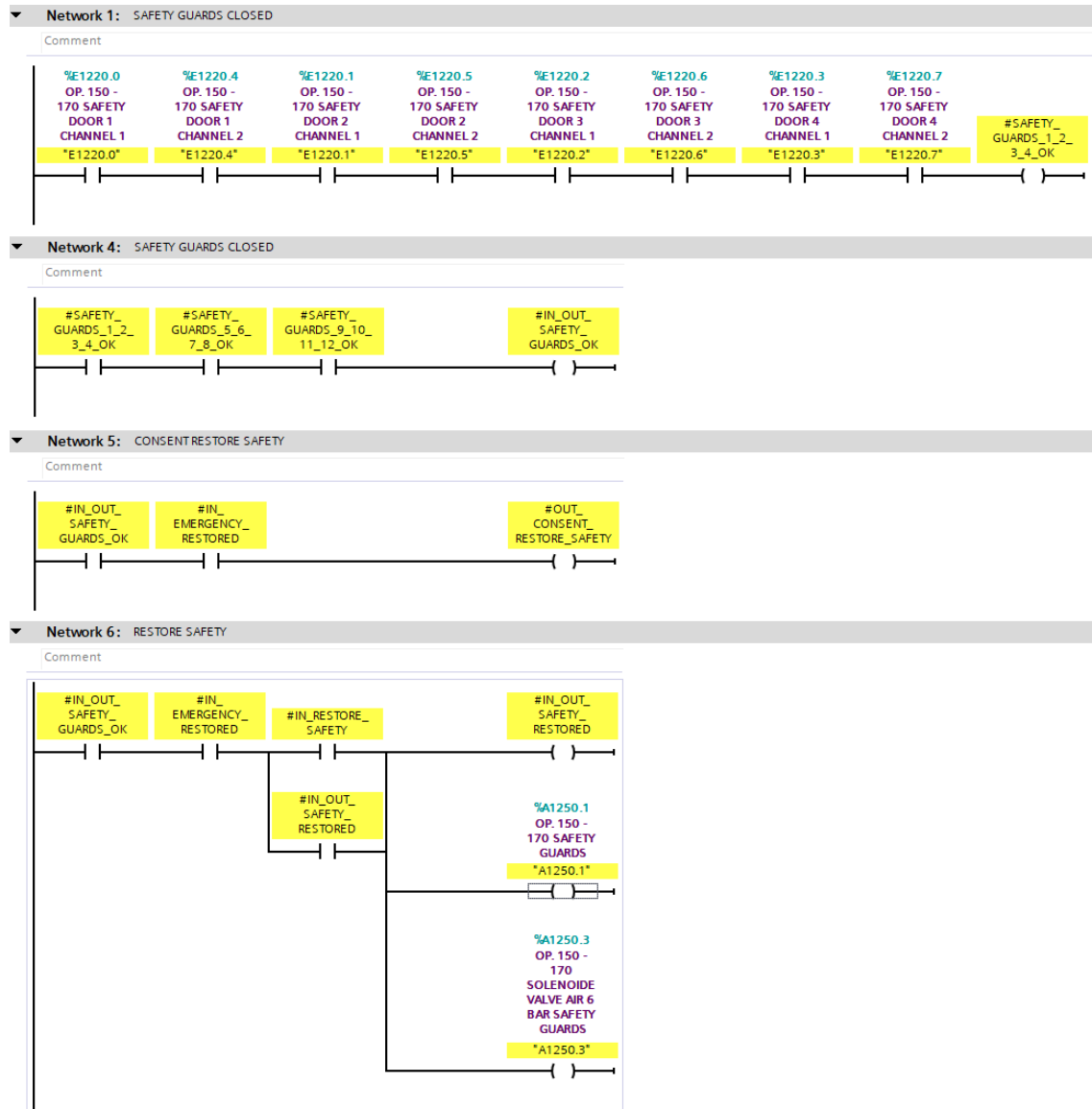


Figure 5.1: Safey Doors Function.

emergency buttons, but is done exactly like the one explained in a moment. These functions have then been included inside the **Main_Safety** program block.

In general a function makes use of different types of variables: there are Input variables coming from outside the function, Input/Output variables that are created inside the function and used as feedback for the function itself and finally there are Output variables that are given to the outside.

The function implemented for the safety doors of the last unit, that groups OP150-160-170, appears like in Fig. 5.1.

The first segment verifies that the first four doors are effectively closed.

If this happens, which means that all the bits are equal to 1, a temporary variable (*SAFETY_GUARDS_1-2-3-4_OK*) is activated. This is done for all the safety guards in the unit in the segments 2 and 3 not showed because perfectly equal to the previous one.

Every safety door has two channels, corresponding to two hardware phisical inputs, for safety reasons.

The same thing has been done for the others eight doors present.

If all the three temporary variable, one for each set of doors, are set, as showed in Segment 4, an Input-Output Variable is activated (*IN_OUT_SAFETY_GUARDS_OK*). This variable is used in Segment 5, togheter with an other Input variable to consent the safety restore, and in Segment 6 to activate another I/O variable and two Output variables.

The first output variable is **A1250.1** and is responsible of commanding the emergency relay associated to the safety guards, while the second is **A1250.3**, that restores the pressure inside the unit, instead.

The Input, Output and Input/Output variables need to be passed into the function or out of the function itself.

This is done by calling (or including) the function inside the Main_Safety Program Block and that appears in Fig. 5.2.

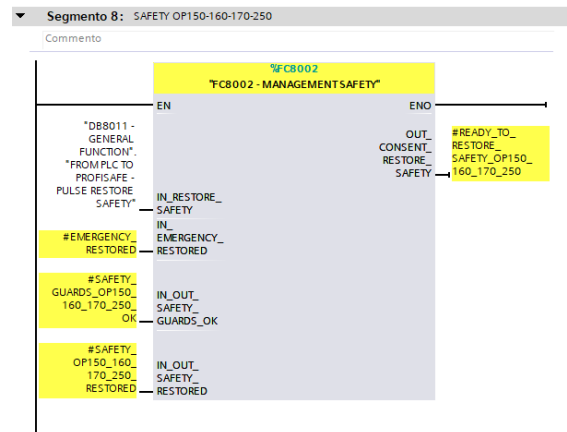


Figure 5.2: FC8002 - Safey Doors Function compiled.

The same procedure has been implemented for the emergency buttons present in the unit.

5.2 Motors

In the previous chapter was described what informations the motors exchange with the PLC, but these informations need to be handled.

The motors have been used in two different ways, depending on how to control their positioning.

In some cases **Absolute Positioning** is required, so that the motor can perform a move in a certain position without being affected by the previous one.

In others **Relative Positioning** is preferable: in this case the next position depends from the previous one.

Let's consider the actual position of the motor is *90* degrees and the desired position is *360* degrees. The motor will automatically move in the direction where the distance is shorter, which is in counter clockwise direction.

This is not always a good situation, especially when the rotation direction of the motor could cause collisions of mechanical parts, but it's convenient in others.

Depending on the applications, one has been preferred to the other.

The general management is however the same and it is described with the following subsection.

5.2.1 General Management

To explain how the management works, the first motor of the unit is considered, which is the one present in OP150 used during the Dowel Pin Insertion.

The first thing to do is acquiring input data from them and, at the same way, send commands data. This has been done using an *AWL* segment code, which is a particular instruction list code developed by Siemens, and that corresponds to a low-level programming which allows more versatility.

In chapter 2 was explained that 6 Input Words and 11 Output Words form the communication telegram between the PLC and the motor. In the same way of the safety and of all the other devices connected to the PLC, also in this case it is necessary to specify which is the starting Input and Output addresses.

For the first motor of the last unit this Address corresponds to 3750.

The input data correspond to the Status Word, the Actual Position, the Actual Speed and in the end the Actual Torque.

The first code line is intended to load the input word (*EW3750*), that starts, as specified, with the address 3750. Once the first word has been read, it is transferred directly inside the specified DB, which in this case is *DB8500 - OP150 Runout 1 Motor*, and that is a DB intended to store the informations on that motor only.

For what concerns the output data, instead, the operation is different. In this case, the element is read from the DB and then it is transferred to the output.

For example with the line 17, the command word is load and transferred to the output word 3750 (*AW3750*) or with line 18 the *TX_DATA_TARGET_POSITION* is load and then transferred to the DInt output *AD3752*. Also the other output words, that are Speed, Acceleration, Deceleration and the two Torque Limit, positive and negative need to be load and transferred.

There is another problem to consider, that is the PLC and the motor drives have

```

1      L      "EW3750"
2      T      %DB8500.DBW0
3      //
4      L      "ED3752"
5      T      "DB8500 - OP150 RUNOUT 1 MOTOR".RX_DATA.ACTUAL_POSITION
6      //
7      L      "ED3756"
8      T      "DB8500 - OP150 RUNOUT 1 MOTOR".RX_DATA.ACTUAL_SPEED
9      //
10     L      "EW3760"
11     T      "DB8500 - OP150 RUNOUT 1 MOTOR".RX_DATA.ACTUAL_TORQUE
12
13
14     CALL "FC8500 - OP150 RUNOUT 1 MOTOR"
15
16     //
17     L      %DB8500.DBW20
18     T      "AW3750"
19     //
20     L      "DB8500 - OP150 RUNOUT 1 MOTOR".TX_DATA.TARGET_POSITION
21     T      "AD3752"
22     //
23     L      "DB8500 - OP150 RUNOUT 1 MOTOR".TX_DATA.SPEED
24     T      "AD3756"
25     //
26     L      "DB8500 - OP150 RUNOUT 1 MOTOR".TX_DATA.ACCELERATION
27     T      "AD3760"
28     //
29     L      "DB8500 - OP150 RUNOUT 1 MOTOR".TX_DATA.DECCELERATION
30     T      "AD3764"
31     //
32     L      "DB8500 - OP150 RUNOUT 1 MOTOR".TX_DATA.TORQUE_LIMIT_POSITIVE
33     T      "AW3768"
34     //
35     L      "DB8500 - OP150 RUNOUT 1 MOTOR".TX_DATA.TORQUE_LIMIT_NEGATIVE
36     T      "AW3770"

```

Figure 5.3: Transfer Motor Input and Output Data.

different endianness. This means that the bytes are swapped, so the booleans of the Status and Control words have been disposed in the following manner to have the signals stored in the correct way (Fig.5.4).

| | | | | | | | | |
|----|---|-------------------------|--------|------|---|--------------------------|--------|------|
| 3 | ▼ | STATUS_WORD | Struct | 0.0 | ▼ | TX_DATA | Struct | 20.0 |
| 4 | ■ | ACTUAL_OPERATING_MODE_1 | Bool | 0.0 | ▼ | COMMAND_WORD | Struct | 20.0 |
| 5 | ■ | ACTUAL_OPERATING_MODE_2 | Bool | 0.1 | ■ | OPERATING_MODE_1 | Bool | 20.0 |
| 6 | ■ | COMMAND_VALUE_ACK | Bool | 0.2 | ■ | OPERATING_MODE_2 | Bool | 20.1 |
| 7 | ■ | MESSAGE | Bool | 0.3 | ■ | RESERVED_1 | Bool | 20.2 |
| 8 | ■ | WARNING | Bool | 0.4 | ■ | RESERVED_2 | Bool | 20.3 |
| 9 | ■ | ERROR | Bool | 0.5 | ■ | IPOSYNCR | Bool | 20.4 |
| 10 | ■ | READY_FOR_OPERATION_1 | Bool | 0.6 | ■ | DRIVE_HALT | Bool | 20.5 |
| 11 | ■ | READY_FOR_OPERATION_2 | Bool | 0.7 | ■ | RESERVED_3 | Bool | 20.6 |
| 12 | ■ | OPERATING_MODE_ACK_1 | Bool | 1.0 | ■ | DRIVE_ON | Bool | 20.7 |
| 13 | ■ | OPERATING_MODE_ACK_2 | Bool | 1.1 | ■ | COMMAND_VALUE_ACCEPTANCE | Bool | 21.0 |
| 14 | ■ | IN_REFERENCE | Bool | 1.2 | ■ | OPERATING_MODE | Bool | 21.1 |
| 15 | ■ | IN_STANDSTILL | Bool | 1.3 | ■ | START_HOMING | Bool | 21.2 |
| 16 | ■ | IN_POSITION | Bool | 1.4 | ■ | RELATIVE_TARGET_POSITION | Bool | 21.3 |
| 17 | ■ | COMMANDE_CHANGE_BIT | Bool | 1.5 | ■ | IMMEDIATE_BLOCK_CHANGE | Bool | 21.4 |
| 18 | ■ | OPERATIVE_MODE_ERROR | Bool | 1.6 | ■ | ERROR_DELETE | Bool | 21.5 |
| 19 | ■ | DRIVE_HALT | Bool | 1.7 | ■ | JOG_FORWARDS | Bool | 21.6 |
| 20 | ■ | ACTUAL_POSITION | Dint | 2.0 | ■ | JOG_BACKWARDS | Bool | 21.7 |
| 21 | ■ | ACTUAL_SPEED | Dint | 6.0 | ■ | TARGET_POSITION | Dint | 22.0 |
| 22 | ■ | ACTUAL_TORQUE | Int | 10.0 | ■ | SPEED | Dint | 26.0 |
| | | | | | ■ | ACCELERATION | Dint | 30.0 |
| | | | | | ■ | DECELERATION | Dint | 34.0 |
| | | | | | ■ | TORQUE_LIMIT_POSITIVE | Int | 38.0 |
| | | | | | ■ | TORQUE_LIMIT_NEGATIVE | Int | 40.0 |

Figure 5.4: Motor Data Block - Input and Output data.

This DB is not optimized, so every information passed to it has an absolute address and it is possible to point it directly during the programming. The address corresponds to the offset, as showed in the picture.

Once the configuration has been completed, a function has been created to manage the motor, containing code segments written with ladder logic.

This function is called inside the previous AWL code in line 14.

There are too many segments inside the function, so only few of them are described to better understand how input and output data are used.

As an example of the Status Word signals management, this segment line shows how the PLC interrogates two bits sent by the drive to understand in which status mode the drive is (Fig. 5.5).

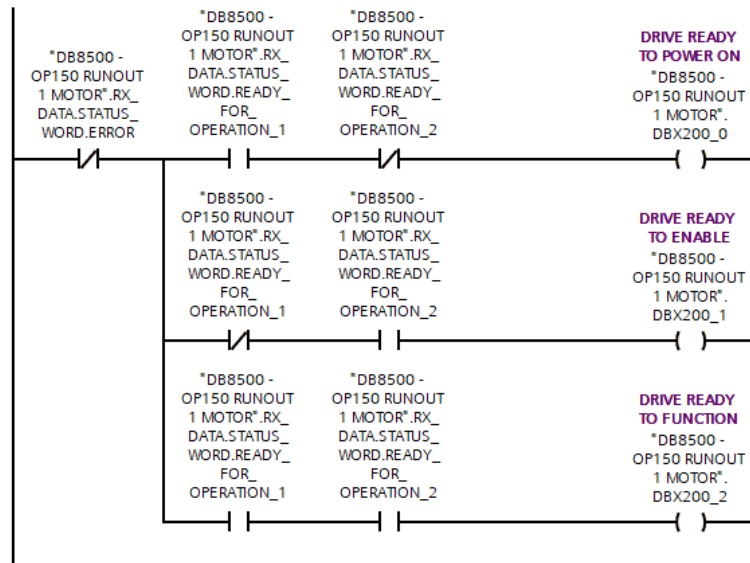


Figure 5.5: Drive - Operating Mode.

Making reference to Fig. 2.4, depending on combination of the two signals, represented by bit 14 and 15, the drive can be in different states:

- **1 - 0**

If the *Ready_for_Operation_1* signal passes from 0 to 1 and *Ready_for_Operation_2* passes from 1 to 0, then the drive is ready to power on

- **0 - 1**

If the *Ready_for_Operation_1* signal passes from 1 to 0 and *Ready_for_Operation_2* from 0 to 1, then the drive is ready enable

- **1 - 1**

If both *Ready_for_Operation_1* and *Ready_for_Operation_2* signals pass from 0 to 1, then the drive is ready for operating

Other segments are used to check if the motor is *In Standstill*, if it is *In Reference*, that means that the drive knows through the encoder the exact position of the motor, and all of the other feedbacks that the drive sends to the PLC. Others, again, are used to pass fundamental parameters for the cycle, such as target position, speed, acceleration and so on.

Inside the same DB there are also elements in which are stored the numerical values

of the parameters just mentioned.

These elements are overwritten during the operation cycles, to change their values depending on the situations (usuually only taget positions and speed are changed).

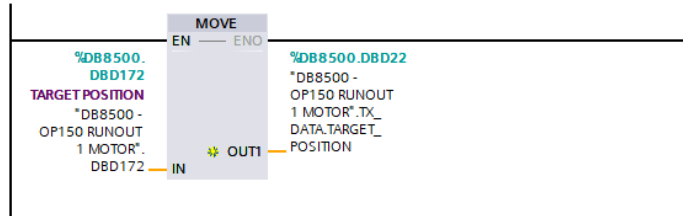


Figure 5.6: Target Position Transfer.

Thanks to the **Move** block implemented by Siemens, it is possible to move a value into a specific variable or DB element.

During the cycle operations the *TARGET POSITION* element, present in the motor DB, is changed depending on the position the motor needs to reach. This value is then moved into the *TX_DATA_TARGET_POSITION* that is directly transferred to the drive thanks to the AWL code showed at the beginning of this section.

This thing is done for all the informations to be passed to the motor, so it happens for the speed, acceleration and so on.

What explained so far it is the general management of the motors. Each motor has its own function, that differs from the other, because each of them need to work in different conditions and with different parameters.

5.2.2 Absolute and Relative Positioning

The main difference regards the calculation of the **Following Error**. This value is used to understand how much the motor is near to the required position and it's represented by a temporary variable given by the absolute value of the difference between the actual position of the motor and the required target position.

If the motor is set to **Absolute Positioning** then the following error is calculated as showed in the next picture (Fig.5.7).

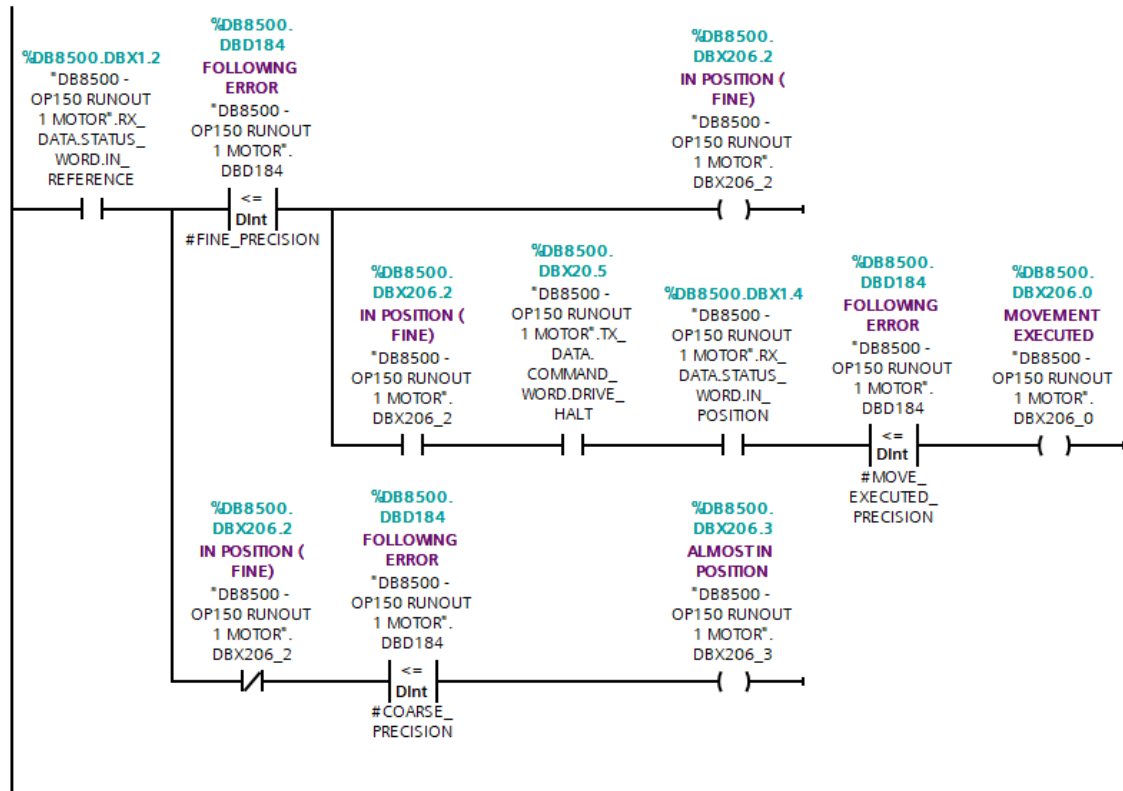


Figure 5.7: Motor Following Error.

Three different positioning windows are used to specify how much the motor is close to the request position:

- **FINE POSITION:** the tolerance is plus or minus 1 degree
- **COARSE POSITION:** the tolerance is plus or minus 5 degrees
- **MOVE EXECUTED POSITION:** the tolerance is plus or minus 0.5 degrees

If the motor is in reference and the following error is less or equal than the *FINE PRECISION*, then the motor is *IN POSITION(FINE)*, that is like saying that the motor position is inside the *FINE POSITION* windows.

If the motor is *IN POSITION(FINE)* then the PLC elaborates the next instructions: if the *DRIVE_HALT* command has been given to the drive (the Halt Drive is a kind

of ‘*Start*’, Fig.2.5), if the motor is *IN POSITION* and if the following error is less or equal then the *MOVE EXECUTED POSITON* then the motor has executed the movement.

Otherwise if he motor is not *IN POSITION(FINE)* the PLC executes the other instruction, that is verifying if the following error is less or equal then the *COARSE POSITION*. If this is so, then the motor is almost in position.

These three outputs information about the motor position are saved inside the motor DB and used during the operation cycles.

The things change if the motor is set to **Relative Positioning**.

As anticipated before the rotation direction of the motor is decided depending on the shorter direction to reach the target position and further operation is done.

If the absolute value of the **following error** is greater than 180 (half revolution of the motor), then:

$$\text{following error} = 360 - \text{following error} \quad (5.1)$$

The rest is the same as the others.

5.3 Robots Management

Two SCARA robots have been used to pick the pieces from the pallets they are placed on and to put them under the stations in which they have to be worked on. Also in this case there is a first general implementation, equal for each one and then a specific function for each of them, that depends on the robot and that is meant to pass them informations to make them work in the proper way.

5.3.1 General Function Block

The first thing done was implementing a **Function Block**, which is different from a normal **Function**, since it has its own instance DB.

However, it is like the function for what concerns the programming, so the same procedure described before for the motors has been done.

A function block is preferable because, unlike the function of the motors previously

described, this one is implemented only for giving the basic commands to the robots, representing the first byte of Fig. 2.6, and for receiving feedbacks from them, always the first byte of Fig. 2.7.

Moreover, it is not necessary to create an apposite DB for each motor, for the function block creates automatically it on its own when the function is called.

Thus, this function block does not transfer all the 32 Input Bytes inside the instance DB, but only the first one, corresponding to the *Status Byte* and does the same for the *Control Byte*, whose individual bits are stored and used inside the function block and then transferred out to send commands. However the same procedure of Fig. 5.3 is utilized. An example of how these input and output signals are used is showed in the next picture (Fig. 5.8).

The first segment (Network 2) is used to give standard commands to the robot. Considering the first instruction executed by the PLC, if all those conditions are verified, a *Command_Start* boolean variable passes from 0 to 1. This variable is loaded and then transferred to the corresponding input boolean of the robot, which is *Start*.

The second segment (Network 12) set static variables inside the instance DB, by acquiring feedback signals coming from the robot.

If the emergency has been restored, the safety guards closed, so that their signals pass from 0 to 1, and there is the *E_Stop_On* signal rising, then this means that the robot has another emergency that is blocking its functioning.

This variable is then reset inside this same function block, once that signal passes from 1 to 0.

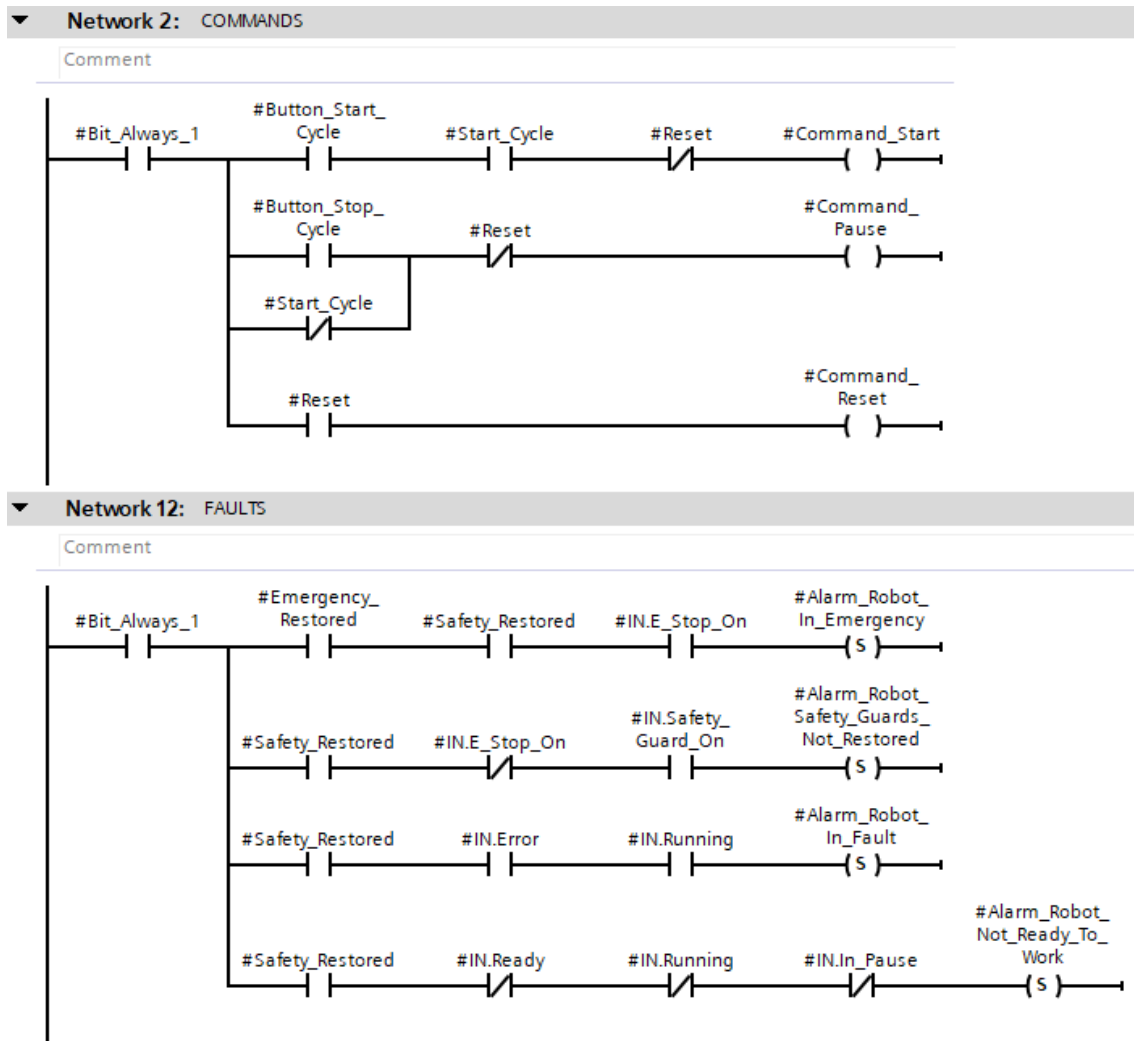


Figure 5.8: Robot Function Block - Commands and Faults.

5.3.2 Robot Programming

Those previous input and output signals are not sufficient. Other informations are required to make the robot move and perform a real working cycle.

The robot cycle has been implemented through its specific programming tool and is managed by its own controller, that is the *RC-700A*.

This software running on the robot controller must be interfaced with the PLC, forming a kind of ‘hand shake’, so that the PLC and the robot controller cannot

work without a mutual consent. However, the master is always the PLC, which communicates the exact function to be performed by the robot, depending on the application and explained later on on this subsection.

First of all it was necessary to take some points, used to define the pick and place positions and the trajectory the robot must follow.

Considering as example the first robot present on OP150: it has two grippers and has to pick and place two different pieces, which have different dimensions.

Thus, several points have been taken for each possible combination of gripper and type of piece.

A total of 21 points have been taken, both unlocking the brakes of the axes or moving the specified axis in jog mode through the robot programming tool to have more accuracy.

To these points have then been assigned specific names, in order to be used for programming the cycle.

The robot program consists of a loop cycle, in which the robot waits the function to be performed and that is passed by the PLC through a *DInt* variable called **Mission_Code**.

```
1 Do
2
3     Select Case nStart
4
5         'Pick and Place on Pallet
6         Case 100
7             Pick_Place_on_Pallet
8
9         'Pick from Pallet
10        Case 101
11            Pick_From_Pallet
12
13        'Place on Pallet
14        Case 112
15            Place_on_Pallet
16
17        'Pick from Pallet
18        Case 201
19            Pick_From_Pallet
20
21        .
22        .
23        .
24
25        'Place on Runout 1
26        Case 1101
27            Place_on_Runout_1
```

```

27
28         'Pick and Place on Runout 2
29         Case 2000
30             Pick_Place_on_Runout_2
31
32         'Pick From Runout 2
33         Case 2002
34             Pick_From_Runout_2
35
36         'Place on Runout 2
37         Case 2101
38             Place_on_Runout_2
39
40         Default
41         CycleSelection
42
43     Send
44
45     Loop
46
47 Fend

```

Once the robot is in the loop cycle, a *Select Case* have been implemented, so that, depending on the mission code passed by the PLC, the robots executes one program rather than another.

The *nStart* corresponds to the function number to be performed and is equal to the value passed through the mission code.

Let's consider the following lines.

```

1  Function Pick_From_Pallet
2
3      Mem_Mission_Status = 1
4
5      nStart = 0
6
7      Print "Pick from pallet"
8
9      'Intake
10     If InW(Mission_Code) = 101 Then
11
12         Mem_Mission_Status = 2
13
14         Pass P_Pick_Index_Pallet_G1_Int :Z(Z-High)
15
16         Wait Sw(DI_Safe_Pos_Index_Pallet) = On And Sw(DI_Free_Station_Index_Pallet) = Off
17
18         Mem_Out_From_Pallet = False
19
20         Mem_Mission_Status = 3
21
22         Move P_Pick_Index_Pallet_G1_Int
23
24         Mem_Mission_Status = 5
25
26         Close_Gripper_1
27
28         Mem_Return_Mission_Code = 101
29

```

```

30         Mem_Mission_Status = 6
31
32         Pass P_Pick_Index_Pallet_G1_Int :Z(Z_High)
33
34         Mem_Out_From_Pallet = True
35
36         'Exhaust
37         Elself InW(Mission_Code) = 201 Then
38         .
39         .
40         .
41         EndIf
42
43         Mem_Mission_Status = 9
44
45 Fend

```

This function, as the name suggests, is that intended for the robot to pick up a piece from the pallet.

The robot enters this function each time the PLC sends one of the two mission codes, *101* for intake or *201* for exhaust (later in this section is explained how). The robot has to move two tipology of pieces, that differ both for dimension and for the station they have to be worked on.

So it is necessary to distinguish the two situations, since also the safety conditions and the trajectory of the robot change.

In case the first mission code is sent, the function executes the following instructions:

- *Line 3* - The robot sends an output, called **Mem_Mission_Status** (2.7). This *DInt* variable is used to inform the PLC at what point of the mission the robot is and its value ranges from 0 to 9 (some are left as spare).
 - 1: Pick in progress
 - 2: Waiting Pick
 - 3: Move down for pick
 - 5: Closing gripper
 - 6: Move up for pick
 - 9: Pick done

In this case the robot sends a mission status equal to 1, to inform that the picking is in progress.

- *Line 14* - *Pass* command is used to perform a ‘Point to Point’ motion. From its previous position, the robot moves to the pick point, in this case *P_Pick_Index_Pallet_G1_Int* .

A point is composed of three coordinates, *X, Y and Z*. In this case the pick point (previously saved with the exact picking coordinates) is modified, so that the robot moves exactly above the pick-up point, without moving to the previously saved *Z* coordinate, that is set to another value, called *Z_High*.

This is done for maintaining a certain cycle time.

The robot is already in position before being able to pick up the piece. When the safety conditions, received as inputs and sent by the PLC, are satisfied, so that the robot can pick, *Line 22* completes the movement thanks to a *Move* command, that moves the robot using a linear interpolation.

- *Line 16* - The robot waits until the safety conditions are reached.
- *Line 28* - This is an echo mission code. It is used as double check to control that the mission has been completed and it is used during the robot cycle (see later) to perform the next operation.

In order for the two systems to communicate with each other, a PLC function is necessary, function that is specific for each robot, since the operations they have to perform are different.

The first part of it is exactly like the motor one, or the function block used for the general management of the robots.

So it is necessary to read the input data and store them inside a specific DB and transfer output data directly from the data block to the robot.

Then it is necessary to create the effective cycle of the robot, which has been done in SCL language.

```

1
2 //
3 // CYCLE CODE 101 – PICK FROM PALLET (INTAKE TIPOLOGY)
4 //
5
6 IF "DB8550 – OP150 ROBOT 7250AZ01".REGISTER.STEP_CYCLE > 0 AND "DB8550 – OP150 ROBOT
  7250AZ01".REGISTER.CYCLE_CODE = 101 THEN
7   CASE "DB8550 – OP150 ROBOT 7250AZ01".REGISTER.STEP_CYCLE OF
8     1: // INITIALIZATION
9       "DB8550 – OP150 ROBOT 7250AZ01".REGISTER.MISSION_CODE := 0;
10
11     IF "DB8550 – OP150 ROBOT 7250AZ01".REGISTER.MISSION_CODE = "DB8550 – OP150 ROBOT
      7250AZ01".REGISTER.MEM_RETURN_MISSION_CODE THEN
12       "DB8550 – OP150 ROBOT 7250AZ01".REGISTER.STEP_CYCLE := 10;
13     END_IF;
14
15     10: // PICK FROM PALLET
16       "DB8550 – OP150 ROBOT 7250AZ01".REGISTER.MISSION_CODE := 101;
17
18     IF "DB8550 – OP150 ROBOT 7250AZ01".REGISTER.MISSION_CODE = "DB8550 – OP150 ROBOT
      7250AZ01".REGISTER.MEM_RETURN_MISSION_CODE THEN
19       "DB8550 – OP150 ROBOT 7250AZ01".REGISTER.STEP_CYCLE := 40;
20     END_IF;
21     .
22     .
23     .
24
25     40: //
26       "DB8550 – OP150 ROBOT 7250AZ01".REGISTER.STEP_CYCLE := 1000;
27
28     1000: // END CYCLE
29       "DB8550 – OP150 ROBOT 7250AZ01".REGISTER.STEP_CYCLE := 0;
30       "DB8550 – OP150 ROBOT 7250AZ01".REGISTER.CYCLE_CODE := 0;
31       "DB8550 – OP150 ROBOT 7250AZ01".REGISTER.MISSION_CODE := 0;
32
33   END_CASE;
34
35 END_IF;

```

The *REGISTER.CYCLE_CODE* corresponds to the mission code to be passed to the robot. However it is called in another way to distinguish it from the value effectively sent to the robot, which is the *REGISTER.MISSION_CODE* instead.

This value is set depending on the conditions.

Let's consider as example the same mission code as before, so the mission code number *101*, corresponding to the pick from pallet for the intake tipology.

This mission code is set if the gripper is open, there is the consent to pick from pallet and most importantly there is a piece present on the pallet.

If these conditions verify, the cycle code is set to *101* and the *REGISTER.STEP_CYCLE* is set to 1.

This last is used to sequence the cycle through a select case.

- *Line 8* - If the step cycle register is equal to 1 (start cycle) , the mission code sent to the robot is set to 0.

- *Line 11* - If the mission code value is equal to the *REGISTER.MEM_RETURN_MISSION_CODE*, received as echo from the robot, the step cycle is updated and set to 10.
- *Line 15* - Now that the step cycle is equal to 10, the mission code is set to 101, that corresponds to the pick from pallet explained before.
Thus, the robot perform all its operations, until the mission code echo (*Line 28* of the previous lines describing the pick function) is set equal to the mission code.
This means that the robot has effectively picked the piece from the pallet.
At this point the step cycle is updated to 40 (spares are left).
- *Line 25* - The step cycle is set equal to 1000, that corresponds to the end of cycle, where all the registers are set equal to 0.

There is a cycle like this for each function programmed inside the robot.

5.4 Piece and Pallet Identification

This particular section is dedicated to the description of how the data structures, containing the informations on the pieces present on the pallets and on the pallets themselves, have been conceived and how these informations are moved throughout the assembly line so that each station knows exactly which operations the piece must be subjected to.

The first module of Fig. 2.1 is an interface module to connect three antennas, whose function is to read/write the pallet tags present on each pallet. These pallet tags are read/write memories used to store informations. However, since their writing cycles are limited, the informations are not stored inside of them.

To solve this problem, the now well known data blocks (DBs), together with appropriate functions, have been used.

5.4.1 Data Blocks Used

The first DB is the **Pallet** one, which is composed of an array of 200 elements, going from 0 to 199. Each element is composed as follows:

| | |
|--------------------|--|
| ID Tag | It is an array of 7 chars to form a string containing the name of the tag present on the pallet |
| Pallet type | It is a <i>DInt</i> variable to distinguish between an <i>Intake</i> pallet and an <i>Exhaust</i> one. A boolean variable would have been sufficient, but an integer variable has been preferred for any possible additions of products |
| Index Data | It is a <i>DInt</i> variable used to store the element number of the Process DB . This variable is used to associate the pallet with the informations on the piece present on it and on the processes it needs to be subjected at |

Table 5.1: Pallet DB.

| DB85 - PALLET | | |
|---------------|---------------|-----------------------------------|
| | Name | Data type |
| 1 | Static | |
| 2 | ▾ | Array["LOWER_BOUND_PALLET".."..." |
| 3 | ▾ _[0] | "INF_PALLET" |
| 4 | ▾ ID_TAG | "ID_TAG" |
| 5 | ▾ ID_TAG | Array[0..7] of Char |
| 6 | ▾ ID_TAG[0] | Char |
| 7 | ▾ ID_TAG[1] | Char |
| 8 | ▾ ID_TAG[2] | Char |
| 9 | ▾ ID_TAG[3] | Char |
| 10 | ▾ ID_TAG[4] | Char |
| 11 | ▾ ID_TAG[5] | Char |
| 12 | ▾ ID_TAG[6] | Char |
| 13 | ▾ ID_TAG[7] | Char |
| 14 | ▾ PALLET_TYPE | Dint |
| 15 | ▾ INDEX_DATA | Dint |
| 16 | ▸ _[1] | "INF_PALLET" |
| 17 | ▸ _[2] | "INF_PALLET" |
| 18 | ▸ _[3] | "INF_PALLET" |
| 19 | ▸ _[4] | "INF_PALLET" |
| 20 | ▸ _[5] | "INF_PALLET" |
| 21 | ▸ _[6] | "INF_PALLET" |

Figure 5.9: Pallet DB - Data Block Structure.

In picture 5.9 is represented how the previous data block has been implemented and looks exactly like the motor data block described before.

In this case, however, each element has exactly the same structure, that has been explained in the table 5.1.

To avoid creating the same structure for each element, the so called **PLC data types** have been used.

Thanks to this functionality it is possible to define a general structure that can be used multiple times in the program, so that it is not necessary anymore to specify the same structure for each of the 200 elements of the data block. Moreover if something changes, it is only necessary to change the data type once.

There are two of them, the first is the *ID_TAG*, which contains an array of 7 chars. The second is the *INF_PALLET* that contains the previous data type, together with the other two elements, that are the *PALLET_TYPE* and the *INDEX_DATA*. The following data blocks have been implemented exactly in the same way.

The **Process DB** is intended for storing all the informations necessary for the processing of the piece. Even in this case there are 200 elements, each of them composed in the following way:

| | |
|-----------------------|---|
| ID | It is an array of 32 chars to form a string containing the piece identification number, generated the first time the piece reaches the first antenna of the assembly line |
| Product type | Same description made for the <i>Pallet type</i> |
| Exclusions | There are as many boolean variables as the number of operations the piece must be subjected to, for a total of 18 booleans. These variables are supposed to exclude some operations. For example, if tests are not required for an <i>Intake Product</i> , the last three booleans, corresponding to OP150, OP160 and OP170, will be equal to 1 |
| Process | It is a <i>DInt</i> variable used to pass the process to be performed and that corresponds to the operation number (explained later) |
| State Process | It is a <i>DInt</i> redundant variable that has the same value as the previous one and that is useful to double check in case of errors |
| Station Reject | It is a <i>DInt</i> variable to indicate the number of station that has produced a reject |
| Reject Type | It is a <i>DInt</i> variable to distinguish all the possible rejects that the assembly line can generate |

Table 5.2: Process DB.

There is another DB to describe before explaining how informations are moved and this last one is the **Configuration DB**.

It is composed of as many elements as the number of Operations, each of them

composed as showed in the next table.

| | |
|--------------------|---|
| Input Code | It is an array of 5 <i>DInt</i> . These array's elements are Consent values that are written to specify which operation to perform. Some of them are left as spares or used to calibrate the stations with appropriate <i>Master</i> pieces. |
| Output Code | It is an array of 5 <i>DInt</i> , whose values are written to indicate the next operation to be performed. |

Table 5.3: Configuration DB.

The input code is essentially the output code of the previous operation. This means that the input code is a sort of authorization to process a piece.

When a piece reaches for example the operation 150, if the input code is equal to 150, the piece is processed, otherwise the piece continues moving untill the right operation is performed.

If the piece is processed, then the operation set the output code, that will be, exactly, the next input code.

When the pallet reaches the first operation, however, needs to be initialized with these codes and all the informations stored have to be reset.

For these reasons other data blocks are necessary: these ones store preset values to initialize the others data blocks described untill now to a preset value.

5.4.2 Informations Transfer

The pallet moves along the line through a conveyor belt untill the piece has to be worked.

At this point the pallet is stopped through a cylinder, called *Index Cylinder* and showed in the next picture (Fig. 5.10), and slightly lifted by another cylinder (unseen since it is under the pallet) untill the pallet is blocked against a specific structure designed to constrain it, so that is possible to read the antenna and understand the operations the piece must undergo.

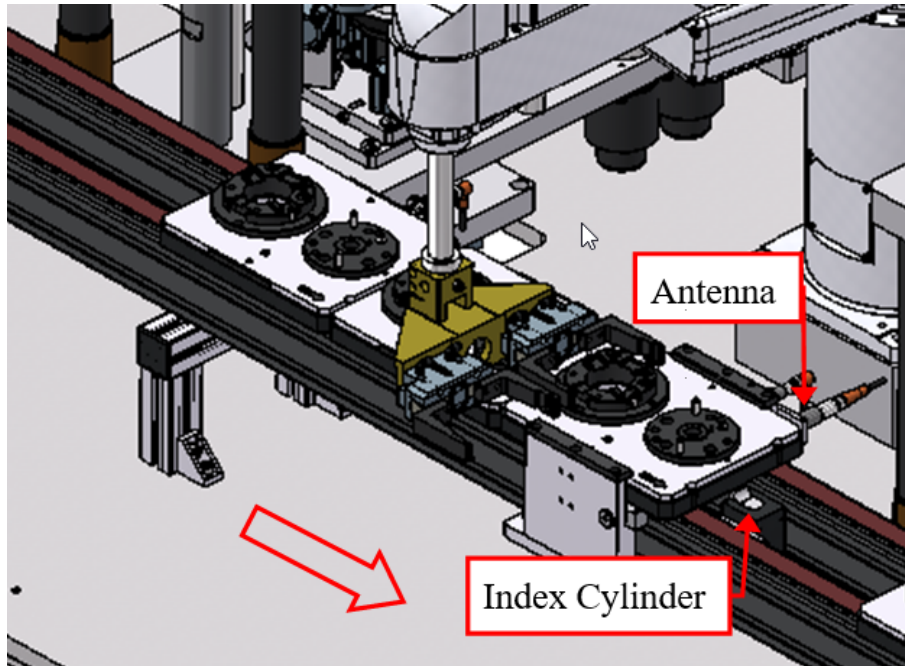


Figure 5.10: Pallet block - Reading the antenna.

Now that the data block have been described, it is possible to explain how the functions to read and write informations have been designed. and the order in which they are called.

- **1 - Read Pallet Tag**

When the pallets arrive in front of any of the antennas distributed along the assembly line, the antennas read the tag identification name, provided by the tag pallet itself through 8 Bytes.

To bypass the writing cycles problem of the pallet tags, this identification name is associated directly to the pallets they are mounted on.

Each pallet has a physical number impressed on it and this number corresponds to the number of the element inside the **Pallet DB** mentioned before. The elements go from 0 to 199, so the pallet number 1 corresponds to the element 1 inside the data block.

The first time a new pallet enters the line (this operation is made only once) the pallet tag must be read, converted and then stored inside the *ID Tag* array of chars (5.1) of the element corresponding to the pallet number.

This is the objective of the next function, written in *SCL* (Structured Control Language) and that, unlike the *AWL*, is a high-level textual programming code based on Pascal.

```
1  // INITIALIZATION VARIABLES
2  //
3
4  #"FC80 - READ TAG BYTE" := -1;
5  #INDEX_PUNT_ARRAY := 0;
6  #FINAL_BYTE := #STARTING_BYTE + 7;
7
8  □FOR #PALLET_BYTE := #STARTING_BYTE TO #FINAL_BYTE BY 1 DO
9
10 □    #TEMP_PALLET_BYTE := PEEK(area := 16#81,
11                                byteOffset := #PALLET_BYTE,
12                                dbNumber := 0);
13
14    #BYTE_ARRAY[#INDEX_PUNT_ARRAY] := #TEMP_PALLET_BYTE;
15    #ID_TAG.ID_TAG[#INDEX_PUNT_ARRAY] := BYTE_TO_CHAR(#TEMP_PALLET_BYTE);
16    #INDEX_PUNT_ARRAY := #INDEX_PUNT_ARRAY + 1;
17
18 □END_FOR;
19
20 □IF (#BYTE_ARRAY[0] = 0) AND
21     (#BYTE_ARRAY[1] = 0) AND
22     (#BYTE_ARRAY[2] = 0) AND
23     (#BYTE_ARRAY[3] = 0) AND
24     (#BYTE_ARRAY[4] = 0) AND
25     (#BYTE_ARRAY[5] = 0) AND
26     (#BYTE_ARRAY[6] = 0) AND
27     (#BYTE_ARRAY[7] = 0) THEN
28
29     #"FC80 - READ TAG BYTE" := -1;
30
31 ELSE
32
33     #"FC80 - READ TAG BYTE" := 1;
34
35 END IF;
```

Figure 5.11: Read Pallet Tag Function.

This function, showed in Fig. 5.11, makes use of indirect addressing, implemented with lines going from 10 to 12 through the *PEEK* instruction, that permits to read a certain memory address.

So it is necessary to specify which memory address to read, in this case an Input Area, indicated by the *16#81*, and the *byteOffset*, that is the byte to read. This function is general, so the *byteOffset* is passed from the outside and corresponds to the considered antenna's starting address.

The last information to specify is the *dbNumber*, that is equal to 0 since it is not a DB the function is reading from.

A simple for cycle is used to convert each byte read into a char.

The function returns also a value, that is *1* if the antenna reads something, or *-1* if it reads nothing, used to call (or not) the other functions. During the manual association it is also necessary to specify the pallet type, contained inside the same data block too.

Once the association has been made, every time a pallet reaches an antenna another function is called, that searches in which of the *n* pallets is stored the same array of chars read from the antenna.

- **2 - Search ID Tag**

If the previous function return value is 1 this other function is called.

This one transforms the incoming array of chars, read from the previous function, into a string and does the same for the arrays of chars of all the elements present in the *Pallet DB*. If the two strings match, the function returns the element number through a *Dint* variable, which corresponds, again, to the pallet number.

Also in this case a return value is generated to continue.

Moreover the pallet number is used to associate the measures effectuated and the results obtained, that are stored in a separate data block, since they need to be sent to a server.

- **3 - Search Free Position inside the Process DB**

This function works similarly to the previous one and it's called only when the pallet enters the line and meets the first antenna.

This one, instead, generates a null string from an array of 32 chars present inside the **Initialization Variable DB** and then compares it with the strings generated from all the *ID* array of chars inside the *Process DB* . If the two strings match, the element number is returned.

That element stores, in addition to the piece identification number, all the informations about the processes the piece must be subjected to.

The element number generated by this function is then moved inside the *Index Data* variable of the pallet.

This is the way a piece is associated to the pallet it is transported from.

• 4 - Process Authorization

This function has been implemented to check whether the piece can be processed or not.

The **Process** variable inside the *Process DB* (5.2) stores a value indicating exactly the process that the piece must undergo, that is written after the previous operation is completed.

Each operation has its processing code associated to it. For example OP150 has value 150, that is exactly equal to the operation number, and so on for all the others.

Inside each element of the *Configuration DB* (there are as many elements as the operations, so there is one element for OP150, one for OP160 and so on) there is the so called *Input Code* array (5.3). The elements of this array store the *Process* values of the corresponding operation. Thus, always considering OP150, the first element of the array is equal to 150, while the others are left as spares.

When the pallet reaches a whatsoever antenna along the line, the **Process Authorization** function checks all the elements inside the *Input Code* array of the element corresponding to the OP the antenna is mounted on, and verifies if one of them (in the previous example there is only the first element of the array different from 0) matches the value of the **Process** value inside the

Process DB. Moreover the function checks if inside the *Process DB* the *Process* is equal to the *State Process* (they have to be equal) and if the piece has not been rejected in some previous operations.

If this happens then the return value of the function is set to *1*, the effective operation cycle starts and an output code is generated, whose value is taken from one of the elements of the **Output Code** array inside the *Configuration DB*. Similarly these values correspond to the suboperations the piece must undergo after the actual process is completed.

When the process is actually completed the output code, generated previously by the *Process Authorization*, overwrites the **Process** and the **State Process** values inside the *Process DB*, and everything starts all over again when the next antenna is intercepted.

5.5 SCADA System

As anticipated before, with an ad-hoc Visual Basic Implementation it is possible to interact directly with the machine.

Thanks to a free library called **LibNoDave** it is possible to read and write informations from and into a whatsoever data block (it is also possible to write input and output directly) created inside the PLC.

This is a very powerful instrument, because the operator can interact with the machine without knowing nothing of PLC programming.

Moreover it is very easy to perform diagnostics or understand what is happening inside the machine.

Before starting with the general implementation of the software, it is worth explaining the data blocks the program interacts with.

- *Alarms and Warnings*: this data block is dedicated to sharing with the operator all the informations regarding the status of the machine.

For example each time a safety guard is opened, or an emergency pushbutton is pressed, a station is on fault, a profinet anomaly occurs, and so on, a boolean variable is **set** inside this specific DB.

The program can read the set variable and the operator knows exactly which one of the doors is opened, so that it's easier for him to perform his job.

This data block is only read.

- *PC Input Int*: it is a read only data block that is meant to send to the program informations that are represented by Integer variables, such as the integer measurement results.
- *PC Input Real*: it is the same as the previous one, but this time all the elements are represented by Real variables.
- *PC Output Int*: this db can be either written or read and is supposed to write all the informations that are represented by Integer values.
- *PC Input Real*: this one is also a read/write DB, but is for reading/writing real data inside the data block.

Thanks to this one it is possible for the operator to choose all the setting parameters for the devices present inside the unit.

It is possible to insert the velocity of the motors, the positions the motors have to reach, the limits the tests must fall into, and so on.

First of all it is necessary to create a communication with the PLC and this has been done creating another specific library to adapt the libnodave one to the PLC programming method used.

```
1 Imports libnodave
2 Public Class PLC_Siemens : Implements IDisposable
3
4 Private Type_Connection_PLC As libnodave.daveOSserialType
5 Private Interface_Connection_PLC As libnodave.daveInterface
6 Private Connection_PLC As libnodave.daveConnection
7 Private Const N_Max_Dati As Integer = 200
8 Private Memory_Buffer(N_Max_Dati - 1) As Byte
9 Private Result As Integer
10 Public Connected As Boolean = False
11 Private Debug As Boolean = False
12 .
13 .
14 .
15
16 Public Sub New(Porta As Integer, Indirizzo As String, Rack As Integer, Slot As Integer, Optional In_Debug As Boolean =
    False)
17 Try
18
19     If In_Debug = False Then
```

```

20         Type_Connection_PLC.rfd = libnodave.openSocket(Porta, Indirizzo)
21         Type_Connection_PLC.wfd = Tipo_Connessione_PLC.rfd
22     .
23     .
24     .
25     Else
26         Debug = True
27         Connesso = True
28
29     End If
30
31     Catch ex As Exception
32         RaiseEvent On_Error("Communication PLC – Connection Error")
33
34     End Try
35
36 End Sub

```

So a new **Costructor** is defined specifying its parameters.

The parameters are the TCP port (*Porta*), the IP address of the PLC (*Indirizzo*), the *Rack*, the *Slot* and an optional variable used to simulate the program without being physically connected to the PLC, setting all to Debug mode.

Togheter with the constructor, also the methods have been designed, that are the actions performed to read and write inside the data blocks and that are called when needed during the program.

Also the methods make use of the libnodave functions and an example is showed in the next lines for what concerns a reading of an Int varibale.

```

1     Public Sub Read(Area_Dati As Area_Dati_Enum, n_DB As Integer, Indice As Integer, n_Dati As Integer, ByRef Dati() As Int16)
2     Try
3         If Debug = False Then
4
5             Dim i As Integer
6             Dim Len As Integer
7             Dim N_Byte As Integer
8             Dim Indice_Lettura As Integer
9             Dim Indice_Scrittura As Integer
10
11             Indice_Lettura = Indice * 2
12             Indice_Scrittura = Indice
13
14             Len = n_Dati * 2
15             N_Byte = Len
16
17             If N_Byte > 0 Then
18
19                 While N_Byte > 0
20
21                     If N_Byte > N_Max_Dati Then
22                         N_Byte = N_Max_Dati
23                     End If
24
25                     Select Case Area_Dati
26
27                         Case Area_Dati_Enum.DB
28                             Result = Connessione_PLC.readBytes(libnodave.daveDB, n_DB, Indice_Lettura, N_Byte,

```

```
29
30         If Result = 0 Then
31             For i = 0 To N.Byte / 2 - 1
32                 Dati(Indice.Scrittura + i) = Connessione.PLC.getS16
33             Next
34         Else
35             RaiseEvent On_Error("Communication PLC – Error Reading DBW Integer")
36         Exit Sub
37     End If
38
39     .
40     .
41     .
42     Case Area_Dati_Enum.A
43         Result = Connessione.PLC.readBytes(libnodave.daveOutputs, 0, Indice.Lettura, N.Byte,
44             Memory_Buffer)
45         If Result = 0 Then
46             For i = 0 To N.Byte / 2 - 1
47                 Dati(Indice.Scrittura + i) = Connessione.PLC.getS16
48             Next
49         Else
50             RaiseEvent On_Error("Communication PLC – Error Reading AW Integer")
51         Exit Sub
52     End If
53 End Sub
```

In this case a select case has been used that is intend to select the memory area that needs to be read. In the previous lines there is an example for reading a datablock (*Area_Dati_DB*) and an output area (*Area_Dati_Enum_A*).

Thus, the properties of the method are used to specify which memory area to read, so that the case is selected, the number of DB (if it is an input or output area this number is set equal to 0), the starting element from where to read (*Indice*), the number of elements to read (*n_Dati*) and finally a buffer to store the results.

Now it is finally possible to import both the two libraries in the program, so that a new communication can be enestablished and informations can be read and written.

```
1 PLC = New OTS.PLC.Siemens(102, "192.168.0.1", 0, 1, Debug)
```

In this case, the data corresponding to the PLC we are dealing with, are passed as parameters to the constructor to create the new object **PLC**, that has:

- Port: 102
- IP: 192.168.0.1
- Rack: 0

- Slot: 1 (this is always 1 for 1500 PLC series)
- Debug: initialized at False

When the connection is established the programm reads automatically all the data blocks described previously in this section:

```

1  Public Sub LetturaDB8300()
2
3      PLC.Read(OTS.PLC_Siemens.Area_Dati_Enum.DB, 8300, 0, 100, DB8300)
4
5  End Sub

```

The previous lines are responsible for reading the first 100 elements of the data block number 8300, that is the one storing all the Alarms and Warnings of the last unit, but there is a method like that for each data block to read.

The read data are stored in the buffer *DB8300* that has been declared as an array of 100 elemets.

Then a graphical interface has been created so that it is possible to see what happens and select what to do.

Let's have a look at the next picture (Fig.5.12).

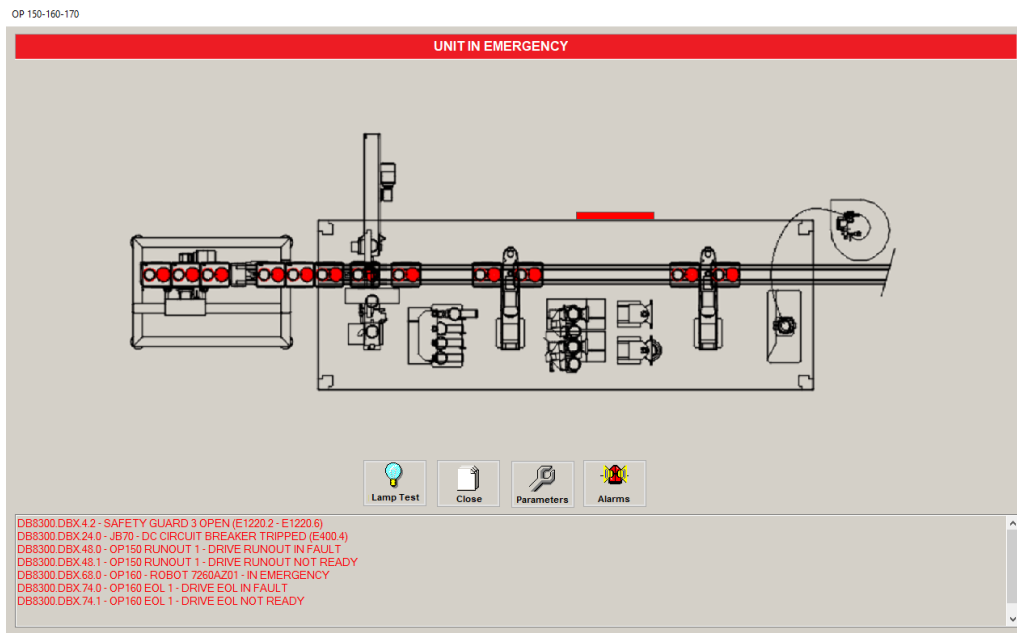


Figure 5.12: Supervision Layout.

Thanks to this interface the operator has always everything under control. The red bar on the upper part of the layout represents the safety guard actually open and the same problem is written also in the lower display bar, together with other problems that are happening.

In this case for example it is showed that a circuit breaker tripped, so that the operator can go on the cabinet and close it (the cabinet is really big, so this is a huge help), and that some of the devices are in fault or not ready.

By clicking on the *Parameter* button, it is possible to insert and modify the parameters for the various operations. In the next picture is represented an example for the first of the two runout motors. In Fig. 5.13 it is possible to specify three

Motor RUNOUT 1


| Motor RUNOUT 1 | | |
|---|--------|---------|
| Description | Value | |
| Speed Auto | 15 | rpm |
| Speed Jog | 5 | rpm |
| Speed Jog (Auto) | 10 | rpm |
| Start Cycle Position | 0 | Degrees |
| | | |
| Display Motor Position | 359.35 | Degrees |
| | | |
|  | | |

Figure 5.13: Motor Parameters Settings.

different speeds and the start cycle position, by clicking on the white space and inserting a value on the keyboard.

```

1  Private Sub lbl_Par_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles _
2      lbl_Speed_Auto.Click, lbl_Speed_Manu.Click, lbl_Speed_Jog.Click, lbl_Start_Pos.Click
3
4      Try
5          .
6          .
7          .
8          If LoginSucceeded Then
9              tastieraNum(sender, sender.tabindex, 8313, True, PermissionParametersReal_DB8313,
              limInfReal_DB8313(sender.tabindex), limSupReal_DB8313(sender.tabindex))
10         End If
11
12     Catch ex As Exception
13
14         MsgBox(ex.Message, vbCritical)
15
16     End Try
17
18 End Sub

```

Thanks to these particular lines it is possible to write directly on the specific element of the DB8313, which is the one where are stored the real values.

tastieraNum is a particular module that permits to send the desired value using the tabindex associated to an object.

Each graphical object created inside Visual has an associated tabindex. So it is simply necessary to set the tabindex of each of the rectangular showed in the picture equal to the element number whose value has to be changed.

Moreover also the actual position of the motor is visible in the grey bar below. This is a only readable information, so it is not possible to modify it.

```

1  lbl_Speed_Auto.Text = Math.Round(DB8313(20), 3)

```

In this case it is only necessary to indicate which element, stored inside the buffer (in this case *DB8313*), to read.

The element number 20 corresponds to the automatic speed of the motor, which is round to the third decimal.

This supervision application is very usefull also for configuring more general parameters, such as the *Input* and *Output* codes stored inside the **Configuration DB** (see 5.4.1).

| OP 150 DOWEL PIN | | OP 150 RUN OUT | | OP 160 EOL | | OP 160 LEAK | | OP 170 | | OP 180 | |
|------------------|-------------|----------------|-------------|------------|-------------|-------------|-------------|------------|-------------|------------|-------------|
| Input Code | Output Code | Input Code | Output Code | Input Code | Output Code | Input Code | Output Code | Input Code | Output Code | Input Code | Output Code |
| 0 | 0 | 150 | 160 | 160 | 165 | 165 | 170 | 170 | 180 | 180 | -1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| OP 150 DOWEL PIN | | OP 150 RUN OUT | | OP 160 EOL | | OP 160 LEAK | | OP 170 | | OP 180 | |
|------------------|-------------|----------------|-------------|------------|-------------|-------------|-------------|------------|-------------|------------|-------------|
| Input Code | Output Code | Input Code | Output Code | Input Code | Output Code | Input Code | Output Code | Input Code | Output Code | Input Code | Output Code |
| 0 | 0 | 150 | 170 | 160 | 165 | 165 | 170 | 170 | 180 | 180 | -1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Figure 5.14: Supervision - normal configuration above, test-skip configuration below

The previous example showed in Fig. 5.14 (image above) is the normal configuration for an intake product.

In this case it is possible to see that the the dowel pin insertion is not performed (it is only necessary for exhaust pieces), so all the input and output codes are set to 0, while the other operations need to be performed.

If one does not want to perform the EOL and Leak tests, it is simply necessary to change the output code of the *Runout* operation. In this case the piece goes directly to the marking station, where it will not be marked, however, since there are not the conditions to mark it as good.

The last operation, which is the 180, and that is not described in this work, is the one intended for unloading the pieces. This one sets the output code equal to -1 and not to 0, since 0 is a value that can be easily set by the PLC if any programming errors exist.

This has been done to better understand the problem, if any of that occurs.

The other elements inside the tables, which are set to 0, are spare codes. These are particularly usefull, for example, in conditions where the stations have to be calibrated with particular pieces, called master pieces, that are, in general, monolithic pieces with tighter tolerances.

Chapter 6

Conclusions

The main objective of this work was to ensure that all the devices were configured and managed to make the various stations, present in the final part of the machine, operate correctly.

Through the various implementations (see chapter 5) and the way in which the operating cycles have been programmed, it was possible to have stations working correctly as specified by the client, respecting the requested cycle time of 9 sec/pz. The bottleneck was mainly represented by the EOL and Leak stations, since these tests require a lot of time to be executed and this is the reason why there is more than one station.

As anticipated, moreover, the final client is still working on internal tests to understand the threshold values and pressures to perform the Leak test.

Thanks to the way in which the informations are stored and transmitted, however, a flexible software has been created, which allows to produce other eventual typologies of pieces (if possible with eventual mechanical retooling) and, most importantly, simply by using the supervision program implemented in *Visual Basic*, it is possible to decide the operations to be performed, excluding those that are not necessary, usefull for example when it is necessary to calibrate a particular station. Moreover the supervision program it is a great help for the operator, that can see the results of the various tests and set the parameters of the devices intended to perform them.

There is still some work to be done, like adding particular pages inside the Supervision program to calculate the efficiency of the unit and add particular requests from the client, such as a *torque vs displacement* graph during the EOL test, to

better understand the frictional effect during all the movement of the rotor, and other parameters to be controlled and displayed.

Thus as future works it is possible to implement these features together with some modifications on the operating cycles, to try to reduce the cycle time a bit more, for example making up for the speed of the motors, speeding up the robots and the various pick and place operations instead.

Bibliography

- [1] Kirk VanGelder, "*Fundamentals of Automotive Technology*", Jones and Bartlett Learning, 24 feb 2017.
- [2] "*Reciprocating ICE components and geometrical properties*", Dipartimento di Energia, POLITECNICO DI TORINO
- [3] W. Bolton, "*Programmable Logic Controllers*" (5th Edition), Newnes.
- [4] PLC Manual', available at <http://www.plcmanual.com/plc-programming>
- [5] "*Study of operation of wagon tippler and side arm charger*", available <https://www.scribd.com/document/321868948/study-of-operation-of-wagon-tippler-and-side-arm-charger>
- [6] SIMATIC ET 200SP - System overview, available at <https://w3.siemens.com/mcms/distributed-io/en/ip20-systems/et-200sp/system-overview/pages/default.aspx>
- [7] CPX Terminal with CPI system, available at https://www.festo.com/net/SupportPortal/Files/404639/CPX-CP_2005-07a_539297i1.pdf
- [8] *Rexroth IndraDrive Drive Controllers MPx-02; MPx-03; MPx-04*, Bosch Rexroth AG.
- [9] Robot Controller RC700 / RC90 Option, available at http://www.epson.com.cn/robots/admin/uploads/product_catalog/files/IO_RC700_RC90_r4.pdf
- [10] Roundtest (Roundform Measuring Instruments), available at <https://www.mitutoyo.co.jp/eng/products/menu/QuickGuide.Roundtest.pdf>
- [11] "*Cam phaser locking pin assembly guide*", available at <https://patents.google.com/patent/US20030196621A1/en>
- [12] User Manual ATEQ D520, available at <https://atequsa.com/wp-content/uploads/2014/07/PREMIER-D-USER-MANUAL-ENGLISH.pdf>
- [13] "*Calculation of Leak amount*", available at [http://www.fukuda-jp.com/en/msz/pdf/Calculation_of_leak_amount\(E\)_01.pdf](http://www.fukuda-jp.com/en/msz/pdf/Calculation_of_leak_amount(E)_01.pdf)

- [14] User Manual ATEQ F620, available at <https://atequsa.com/wp-content/uploads/2014/07/PRIMUS-USER-MANUAL-ENGLISH.pdf>