Motion control of 3 D Cartesian robots for different Mechanical applications using the CoDeSys SoftMotion in PLC.

Supervisor:
Prof. Francesco Castelli Dezza

Presented By:-
Vignesh Nagarajan
Matricola 836727

Academic Year: 2015-16
Abstract

Motion control is a sub-field of automation, encompassing the systems or sub-systems involved in moving parts of machines in a controlled manner. The main components commonly include a motion controller, an energy amplifier, and one or more prime movers or actuators. Motion control may be open loop or closed loop. In open loop systems, the controller sends a command through the amplifier to the prime mover or actuator, and does not know if the desired motion was actually achieved. Typical systems include stepper motors. For the controls with more precision, a measuring device may be added to the system (usually near the end motion). When the measurement is converted to a signal that is sent back to the controller, and the controller compensates for any error, it becomes a closed loop System.

The whole package puts a simple approach to the powerful IEC language at the disposal of the PLC programmer. The applications like “Gasketing” involves a wide variety of seals or packing used between matched machine parts or around pipe joints to prevent the escape of a gas or fluid. Gaskets prevent leakage of liquids or gases by forming impervious barriers between mating surfaces. And for such applications, as the precision plays a wider role in the implementation. The entire program logic is treated in the PLC program and just the pure motion information in the library blocks are processed using the SoftMotion coding and implementation of the Gcodes directly from the CoDeSys environment without the use of CNC machines but using the bus interface to communicate with the drivers driving the motors or the end effectors.

This work involves the motion control of the Cartesian Robots for the applications like Gasketing using SoftMotion technique in PLC. The “SoftMotion” is a part of Controlled Development System (CoDeSys) and the functionality is used for realizing movements from a simple single axis CAMs to complex motions in several dimensions of the development environment. Those applications where sequence and process control or auxiliary functions play a key role alongside the motion functions are ideal for use with SoftMotion (as opposed to those that are exclusively concerned with motion functions). With the implementation of the SoftMotion technique, this work shows the economic efficiency of the system for performing the same task as the NC machine with the same level of precision, just by using the PC of a user, a drive controller and a dedicated HMI.
ACKNOWLEDGEMENT

I would first like to thank my thesis advisor Prof Francesco Castelli Dezza for his consistent feedback and guidance on my work, and steer me towards completion of the work successfully. I would like to express my deepest gratitude towards him for offering an instructive guidance all the time.

I would also like to thank Ing. Alessandro Mancini, General Manager, Automata SPA, without whose help and support, I wouldn’t have got such a wonderful learning opportunity in the premises of the company. I am also thankful to Ing.Giorgio Duni and all other staff members of the company who constantly helped me to gain knowledge and had always been a motivation for the completion of my work.

I would thank the Dean of Automation and control Prof Paolo Rocco, for this program which had been a continuous encouragement in the span of two years.

Finally, I would thank my parents, and all my friends who had been a constant source of encouragement and motivation throughout my career.
<table>
<thead>
<tr>
<th>Sl. No</th>
<th>Description</th>
<th>Pg. no</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1</td>
<td>NC/CNC Systems</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>Configuration of a CNC system</td>
<td>4</td>
</tr>
<tr>
<td>1.3</td>
<td>Serial Communication Architecture</td>
<td>5</td>
</tr>
<tr>
<td>1.4</td>
<td>Limitations of the NC System</td>
<td>6</td>
</tr>
<tr>
<td>1.5</td>
<td>Soft PLC</td>
<td>6</td>
</tr>
<tr>
<td>1.6</td>
<td>HMI</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td><strong>Description of Work</strong></td>
<td>10</td>
</tr>
<tr>
<td>2.1</td>
<td>Interfaces</td>
<td>14</td>
</tr>
<tr>
<td>2.1.1</td>
<td>Sercos</td>
<td>14</td>
</tr>
<tr>
<td>2.1.2</td>
<td>EtherCAT</td>
<td>15</td>
</tr>
<tr>
<td>2.2</td>
<td>Model</td>
<td>16</td>
</tr>
<tr>
<td>2.2.1</td>
<td>The State Diagram</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td><strong>Implementation Methods</strong></td>
<td>19</td>
</tr>
<tr>
<td>3.1</td>
<td>FB Interface</td>
<td>19</td>
</tr>
<tr>
<td>3.2</td>
<td>SoftMotion</td>
<td>23</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Virtual visualization of drives</td>
<td>27</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Motor configuration and drive interface with communication bus</td>
<td>28</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Position Control</td>
<td>31</td>
</tr>
<tr>
<td>3.3</td>
<td>The SoftMotion drive Interface</td>
<td>32</td>
</tr>
<tr>
<td>3.4</td>
<td>Error Handlings</td>
<td>33</td>
</tr>
<tr>
<td>4</td>
<td><strong>Analysis of SoftPLC and SoftMotion library</strong></td>
<td>35</td>
</tr>
<tr>
<td>4.1</td>
<td>Implementation of Gasketing using the SoftMotion technique</td>
<td>35</td>
</tr>
<tr>
<td>-----</td>
<td>----------------------------------------------------------</td>
<td>----</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Summary</td>
<td>35</td>
</tr>
<tr>
<td>4.2</td>
<td>SMC_Interpolator</td>
<td>37</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Inputs of the Function Block</td>
<td>38</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Outputs of the Function Block</td>
<td>41</td>
</tr>
<tr>
<td>4.3</td>
<td>Control axis Function Block</td>
<td>43</td>
</tr>
<tr>
<td>4.3.1</td>
<td>SMC_ControlAxisByPos</td>
<td>43</td>
</tr>
<tr>
<td>4.3.1.1</td>
<td>Inputs to the Function Block</td>
<td>43</td>
</tr>
<tr>
<td>4.3.1.2</td>
<td>Outputs of the function block</td>
<td>44</td>
</tr>
<tr>
<td>4.4</td>
<td>Homing</td>
<td>44</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Inputs of the module</td>
<td>45</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Outputs of the module</td>
<td>47</td>
</tr>
<tr>
<td>4.5</td>
<td>Drive Interface : create PLC application for drives</td>
<td>48</td>
</tr>
<tr>
<td>4.6</td>
<td>Direct creation of OutQueue</td>
<td>51</td>
</tr>
<tr>
<td>4.7</td>
<td>Dynamic SoftMotion programming</td>
<td>52</td>
</tr>
<tr>
<td>4.8</td>
<td>Program Code</td>
<td>53</td>
</tr>
<tr>
<td>4.9</td>
<td>Importing the DXF</td>
<td>58</td>
</tr>
<tr>
<td>4.10</td>
<td>Calculations</td>
<td>63</td>
</tr>
<tr>
<td><strong>5</strong></td>
<td><strong>Real-time Implementation</strong></td>
<td><strong>67</strong></td>
</tr>
<tr>
<td></td>
<td>Analysis and conclusion</td>
<td>71</td>
</tr>
<tr>
<td></td>
<td>Appendix</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>References</td>
<td>73</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Fig No.</th>
<th>Description</th>
<th>Page no.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Development of Motion control Field</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>CNC working diagram in a closed loop</td>
<td>3</td>
</tr>
<tr>
<td>1.3</td>
<td>Configuration of a CNC</td>
<td>4</td>
</tr>
<tr>
<td>1.4</td>
<td>Serial Communication architecture</td>
<td>5</td>
</tr>
<tr>
<td>1.5</td>
<td>HMI network representation in SoftPLC</td>
<td>8</td>
</tr>
<tr>
<td>2.1</td>
<td>Major components architecture</td>
<td>10</td>
</tr>
<tr>
<td>2.2</td>
<td>Configuration of a PAC</td>
<td>11</td>
</tr>
<tr>
<td>2.3</td>
<td>Functional description of a SMC</td>
<td>13</td>
</tr>
<tr>
<td>2.4</td>
<td>The State Diagram</td>
<td>18</td>
</tr>
<tr>
<td>3.1</td>
<td>Execute/Done style FBs</td>
<td>22</td>
</tr>
<tr>
<td>3.2</td>
<td>Implementation structure of SoftMotion in CoDeSys</td>
<td>25</td>
</tr>
<tr>
<td>3.3</td>
<td>Sample HMI for Application</td>
<td>26</td>
</tr>
<tr>
<td>3.4</td>
<td>Sample HMI for Gasketing Application page</td>
<td>26</td>
</tr>
<tr>
<td>3.5</td>
<td>Modbus address generator in Ms Excel</td>
<td>27</td>
</tr>
<tr>
<td>3.6</td>
<td>Virtual visualization for observing the movement in axis</td>
<td>28</td>
</tr>
<tr>
<td>3.7</td>
<td>Concept of a servo system</td>
<td>29</td>
</tr>
<tr>
<td>3.8</td>
<td>Position control using the pulse signal</td>
<td>30</td>
</tr>
<tr>
<td>3.9</td>
<td>Position control diagram of the motor</td>
<td>31</td>
</tr>
<tr>
<td>3.10</td>
<td>Data exchange between structures and hardware</td>
<td>32</td>
</tr>
<tr>
<td>3.11</td>
<td>Function Block with centralized error handling</td>
<td>33</td>
</tr>
<tr>
<td>3.12</td>
<td>Function Block with decentralized error handling</td>
<td>34</td>
</tr>
<tr>
<td>4.1</td>
<td>Gcode generation for the requested motion</td>
<td>36</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>4.2</td>
<td>Assignment of Gcode to SMC interpolator and configuring the motion</td>
<td>37</td>
</tr>
<tr>
<td>4.3</td>
<td>Sigmoidal and Trapezoidal velocity profile</td>
<td>39</td>
</tr>
<tr>
<td>4.4</td>
<td>Plot of torque v/s speed to determine maximum speed of the motor</td>
<td>64</td>
</tr>
<tr>
<td>4.5</td>
<td>Control diagram of position, inputting the calculated parameters in the loop for the set and actual positions</td>
<td>65</td>
</tr>
<tr>
<td>5.1</td>
<td>3D Cartesian robot where the application was implemented</td>
<td>69</td>
</tr>
<tr>
<td>5.2</td>
<td>Z Axis and end effector path movement</td>
<td>70</td>
</tr>
</tbody>
</table>
List of Tables

<table>
<thead>
<tr>
<th>Table no.</th>
<th>Description</th>
<th>Page no.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Characteristics of Stepper motor</td>
<td>12</td>
</tr>
<tr>
<td>3.1</td>
<td>FB Interface</td>
<td>19</td>
</tr>
<tr>
<td>4.1</td>
<td>Interpolator Status</td>
<td>42</td>
</tr>
</tbody>
</table>
CHAPTER 1. INTRODUCTION

It is believed that the engineers who were mainly trained in Power Electronics have developed the area of Motion Control. This is a result of applying control theory to power electronics. Currently it is a fairly matured field, which is almost three decades old. The research on application of microprocessors for electrical drive control in 1970’s laid the foundation stone for the Motion Control area. The historical development of the Motion Control field is shown in Figure 1.1.

![Figure 1.1: Development of the Motion Control field](image)

Though initiated by power electronic engineers, current status of motion control is such that conventionally trained power electronic engineers will probably require following complementary courses in motion control to grasp the development in this area. Motion Control, sometimes also referred as “Servo control” or "robotics" is used in industrial processes to move a specific load in a controlled fashion. These systems can use either pneumatic, hydraulic, or
electromechanical actuation technology. The choice of the actuator type (i.e. the device that provides the energy to move the load) is based on power, speed, precision, and cost requirements. Electromechanical systems are typically used in high precision, low to medium power, and high-speed applications. These systems are flexible, efficient, and cost-effective. Motors are the actuators used in electromechanical systems. Through the interaction of electromagnetic fields, they generate power. These motors provide either rotary or linear motion.

The Aim of this Thesis is to design and develop the system based on the soft PLC for the application of deployment of silicon on the edge of an object (Gasketing). The main objective is to follow the Soft PLC technique instead of the classical technique of NC systems where the machine tools or the robots for the movements are controlled using the computer.

1.1 **NC/CNC SYSTEMS**

The NC, and later CNC, allowed for tremendous increases in productivity for machine tools because the machines could be run automatically without requiring constant attention from their operator. Before the advent of such automation, there was a lesser automation opportunity in the form of hydraulic tracer systems. Such systems used hydraulics to cause the cutting tools of a lathe or mill to follow a template. Under CNC Machining, machine tools function through numerical control. A computer program is customized for an object and the machines are programmed with CNC machining language (called G-code) that essentially controls all features like feed rate, coordination, location and speeds.

With CNC machining, the computer can control exact positioning and velocity. CNC machining is used in manufacturing both metal and plastic parts. First a CAD drawing is created (either 2D or 3D), and then a code is created that the CNC machine will understand. The program is loaded and finally an operator runs a test of the program to ensure there are no problems.

CNC machines can have several axes of movement, and these movements can be either linear or rotary. Many machines have both types. Cutout machines like lasers or waterjets generally have just two linear axes, X and Y. Milling machines usually have at least three, X, Y, and Z, and can have more rotary axes. A five axis milling machine is one that has three linear axes
and two rotaries, allowing the cutter to operate in a full 180° hemisphere and sometimes more. Five axis lasers exist as well. A robot arm might have more than five axes.

![CNC working diagram in closed loop](image)

**Fig 1.2 CNC working diagram in closed loop [10]**

The CNC systems are constructed with a NC unit integrated with a Programmable Logic Controller (PLC) and sometimes with an additional external PLC (non-integrated). The NC controls the spindle movement and the speeds and feeds in machining. It calculates the traversing path of the axes as defined by the inputs.

The PLC controls the peripheral actuating elements of the machine such as solenoids, motors, etc. Enabled together, NC and PLC help the machine to operate automatically as per the need of the user. The positioning and the part accuracy depends on the CNC system’s computer control algorithms, the system resolution and the basic mechanical machine accuracy. Control algorithms may cause errors while computing, which will reflect during the contouring, though they are very negligible.
Computer Numeric Control (CNC) is a specialized and versatile form of Soft Automation and its applications cover many kinds although it was initially developed to control the motion and operation of machine tools.

Computer Numeric Control may be considered to be a means of operating a machine though the use of discrete numerical values fed into the machine where the required ‘input’ technical information is stored on a kind of input media. The machine follows a predetermined sequence of machining operations at a predetermined speeds necessary to produce a work piece of right characteristics and thus according to completely predictable results. A different product can be produced through programming and a low quantity production run of different products is justified.

1.2 CONFIGURATION OF A CNC SYSTEM

![Diagram of CNC System](image)

Fig 1.3 Configuration of a CNC [10]
A CNC system consists of six major devices:

a) Input Device  
b) Machine control unit  
c) Machine tool  
d) Driving System  
e) Feedback Devices  
f) Display Unit

The input devices can be of many types based on the memory, ranging from floppy disks to the serial communications being the major ones.

Serial communications help in the transfer of the data between a computer and a CNC machine tool through a communication port.

The architecture of the communication system and the port can be shown in the figure below.

1.3 Serial communication architecture:

![Serial Communication Architecture](image)
The communication from the CNC to the PC is the status of the machine and the one from the PC to CNC is understood as CNC Part programs. These CNC part programs are the data inputs that are given to the CNC in the format of the machine understandable codes, also called as “Gcodes” for the machines to execute the codes for a particular application.

1.4 Limitations of the NC Systems

1. CNC machines are more expensive than manually operated machines, although costs are slowly coming down.
2. Most CNC controllers only understand straight line movements and circular arcs.
3. The speed at which the machine controller can receive and process the incoming data, transmit commands to the drive system, and monitor the machine’s speed and position is critical.

1.5 SoftPLC

A SoftPLC is a high-performance, feature-rich Programmable Automation Controller (PAC), that merges the functions of a proprietary PLC with the benefits of an open architecture design and the power of computers. SoftPLC combine the functions of:

- PLC’s
- Data Loggers
- Communication gateways
- Embedded Computers

At the heart of each system is the SoftPLC Control Software, which is advanced embedded software technology (the "firmware"). With this common core, all SoftPLCs have the same features/functions, use the same development software, and user applications are able to run on different controller hardware models.
All SoftPLC controller CPU’s provide the same rich set of features. Some of these are:

- Reliable solution for both process and machine control.
- Deterministic, high speed program execution.
- "Unlimited" memory for user logic and data table.
- Data logging.
- Fully documented applications, all documentation resides in the SoftPLC controller.
- Interfaces to I/O systems and other devices from hundreds of vendors.
- Supports numerous communication protocols – Sercos, ethernet, serial and industrial networks.
- Compatible with all HMI products, including Visual Basic/C# programs.
- Supports user add-on functions and drivers written in C++.

These SoftPLCs play a major role in widely replacing the CNC machines by involving all the features of the NC to the computer and eases the process of integration of the codes for the running of the application to the drivers through the serial buses. Hence, the SoftPLCs are helpful in reducing the costs and maintains the economic stability of any application process through the software.
1.6 Human Machine Interface (HMI)

The HMI acts as an interface medium between the user and the access for the application runtime. This HMI can communicate using the bus network through the software using the computer and also can be used as a standalone machine in the application platform. The PC can either be part of the controller network or directly connected to the controller.

Depending on the application required by the user, the UI is designed using the Win Mach Lite (WML) software, linking the various required parameters in the application and integrating this into the HMI. The application designed in WML is downloaded into the standalone touch panel (HMI) using the Ethernet and hence can be launched by the user at his requirements and ease.

Fig 1.5 HMI network representation in SoftPLC
The main features of these HMIs are:

- Portable display of visualization.
- Data exchange between PC using the server of the software.
- Display of several controllers in one device.
- Distributed visualization.
CHAPTER 2
DESCRIPTION OF THE WORK

The main aim of the work is to implement the soft motion techniques for the motion control applications using the Programmable Automation Controllers (PAC). The main focus will be for the Gasketing, which involves the following of a path or any trajectory for the sealing of the machine parts to prevent the escape of the gases/fluids.

Fig. 2.1 Major components Architecture

For these works to be performed using the robots, there are prior tests carried out using a panel consisting of the motors, their drivers and a controller. All the controls for the motion of the motors are inputted using the HMI to which the whole system is interfaced.
The major devices and components involved in the architecture are assembled as shown in the fig 2.1.

The **HMI panel** is where the user gets an access to select the type of the application required to be executed by the robot or the tasks that has to be performed by the motor. The user can also input the various important parameters required to drive the motors to the desired position; like the counts per revolution, the desired speed depending on the maximum speed of the motor, the velocity counts of the motor etc.

The HMI is connected to the **Programmable Automation Controller (PAC)**, a device family which combines the advantages of PC-based controls with the typical features of PLCs and motion controller. The basic version is equipped with an Intel Atom dual-core CPU.

![Fig 2.2 Configuration of a Programmable Automation Controller [7]](image)

Despite its extremely compact size, all A2-PAC base units offer a variety of communication interfaces. In addition to the typical PC interfaces such as Ethernet, USB, HDMI additional interfaces for use as a machine or system control are available. The real-time Ethernet interface supports Sercos and EtherCAT, each as a master or slave. Also already onboard available are CAN and digital I/Os. These interfaces can either be addressed by the system CPU over PCIe or a second onboard CPU. In
this configuration the data exchange with the system CPU is done over a fast dual-port RAM interface. The A2-PAC base units are programmed with CODESYS IEC 61131-3 or in C/C++.

The number of the stepper motor controllers (SMC) used depends on the number of motors which in turn depends on the number of axis of movement required in the application. In this case, assuming that there are three axis of movement in X, Y and Z, we have three SMCs. These controllers are linked to the PAC for receiving the commands for the positioning of the motors. The configuration of the motion of the motors, parametric adjustments and the other calculations are done using the CoDeSys v3.

The stepper motors that are used for the experimental purposes for the movement of the axis are the Nanotec AS5918S2804-E with the following characteristics.

Nanotec AS5918S2804-E Stepper motor Configuration:

<table>
<thead>
<tr>
<th>Motor Type</th>
<th>AS5918S2804-E</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current</td>
<td>2.83</td>
<td>(A/Phase)</td>
</tr>
<tr>
<td>Holding Torque</td>
<td>85</td>
<td>(N-cm)</td>
</tr>
<tr>
<td>Resistance</td>
<td>0.75</td>
<td>(ohm/phase)</td>
</tr>
<tr>
<td>Inductance</td>
<td>2.6</td>
<td>(mH)</td>
</tr>
<tr>
<td>Rotor Inertia</td>
<td>230</td>
<td>(g cm²)</td>
</tr>
<tr>
<td>Weight</td>
<td>0.30</td>
<td>(Kg)</td>
</tr>
<tr>
<td>Length</td>
<td>77</td>
<td>(mm)</td>
</tr>
</tbody>
</table>

Table 2.1 Characteristics of the stepper motor.
The implementation of the application is done by initially configuring the involved components in the following manner.

**Fig 2.3 Functional description of a Stepper motor controller [7]**

The SMC3 is a stepper motor controller with real-time Ethernet interface (Sercos or EtherCAT), integrated power stage for one 2-phase stepper motor, digital I/Os and incremental encoder input. The device can be operated with incremental encoder (Closed-Loop) or without incremental encoder (Open-Loop). Depending on the selected real-time Ethernet protocol and the respective drive profile velocity, position and positioning mode are supported.

Features of SMC:

- Real-time Ethernet interface with 2 x RJ45 connectors.
- Sercos FSP-Drive profile or EtherCAT CoE with DS-402 profile.
- 5 diagnosis LEDs.
- Power stage for a 2-phase stepper motor with max. 6 A per phase at max. 48 VDC supply.
• 4 digital inputs.
• 4 digital outputs.
• 1 encoder input for incremental encoder (A, /A, B, /B, N, /N).
• 5 VDC and 24 VDC output for encoder supply.

2.1 Interfaces:

IEC 61131-3 is the third part (of 10) of the open international standard IEC 61131 for Programmable Logic Controllers, and was first published in December 1993 by the IEC. Part 3 of IEC 61131 deals with basic software architecture and programming languages of the control program within PLC. It defines two graphical and two textual programming language standards:

• Ladder Diagram (LD), graphical.
• Function block diagram (FBD), graphical.
• Structured Text(ST), textual.
• Instruction List (IL), textual.
• Sequential Flow Chart(SFC), has elements to organize programs for sequential and Parallel control processing.

2.1.1 Sercos

In the field of Industrial Control Systems, the interfacing of various control components must provide means to coordinate the signals and commands sent between control modules. While tight coordination is desirable for discrete inputs and outputs, it is especially important in motion controls, where directing the movement of individual axis of motion must be precisely coordinated so that the motion of the entire system follows a desired path. Types of equipment requiring such coordination are, for example, metal cutting machine tools, metal forming equipment, assembly machinery, packaging machinery, robotics, printing machinery and material handling equipment. The Sercos (serial real-time communication system) interface is a globally standardized open digital interface for the communication between industrial controls, motion devices (drives) and input output devices (I/O). It is classified as standard IEC 61491 and EN 61491. The Sercos interface is designed
to provide hard real-time, high performance communications between industrial motion controls and digital servo drives.

Important features of the Sercos interface include:

- Collision-free communication through the use of a time-slot mechanism.
- Highly efficient communication protocol (little overhead).
- Extremely low telegram jitter (specified at less than 1 microsecond, in practice as low as 35 nanoseconds).
- Highly developed standardized profiles agreed upon by multi-vendor technical working groups for dependable interoperability of devices from different manufacturers.
- Ability to control, for example, 70 axes of motion at an update of 250 microseconds for each and every drive (Sercos III).

2.1.2 EtherCAT

EtherCAT - Ethernet for Control Automation Technology - is an Ethernet-based fieldbus system, invented by Beckhoff Automation. The protocol is standardized in IEC 61158 and is suitable for both hard and soft real-time requirements in automation technology.

The goal during development of EtherCAT was to apply Ethernet for automation applications requiring short data update times (also called cycle times; \( \leq 100 \mu s \)) with low communication jitter (for precise synchronization purposes; \( \leq 1 \mu s \)) and reduced hardware costs.

With EtherCAT, the standard Ethernet packet or frame (according to IEEE 802.3) is no longer received, interpreted, and copied as process data at every node. The EtherCAT slave devices read the data addressed to them while the telegram passes through the device, processing data "on the fly". Similarly, input data are inserted while the telegram passes through. A frame is not completely received before being processed; instead processing starts as soon as possible. Sending is also conducted with a minimum delay of small bit times. Typically the entire network can be addressed with just one frame.

Features of EtherCAT:

- High Performance
  - EtherCAT is the fastest Industrial Ethernet technology
- Flexible Topology
  - Benefit not only for widely distributed applications
- Ease of Use
  - Easy configuration and maintenance
- Low Cost
  - Inexpensive implementation & infrastructure
- Functional Safety
  - Safety communication integrated
- Product Variety
  - Great variety of available EtherCAT products

2.2 Model

The Function Block (FB) library is designed for the purpose of controlling axes via the language elements consistent with those defined in the IEC 61131-3 standard. It was decided by the task force that it would not be practical to encapsulate all the aspects of one axis into only one function block. The retained solution is to provide a set of command oriented function blocks that have a reference to the axis, e.g. the abstract data type ‘Axis’, which offers flexibility, ease of use and reusability.

Implementations based on IEC 61131-3 (for instance via Function Blocks and SFC) will be focused towards the interface (look-and-feel / ‘proxy’) of the Function Blocks. This specification does not define the internal operation of the Function Blocks.

2.2.1 The State Diagram

Function Blocks are «simultaneously» activated. This combination of motion profiles is useful in building a more complicated profile or to handle exceptions within a program. (In real implementations there may be additional states at a lower level defined).

The basic rule is that motion commands are always taken sequentially, even if the PLC had the capability of real parallel processing. These commands act on the axis’ state diagram.

The axis is always in one of the defined states (see diagram below). Any motion command that causes a transition changes the state of the axis and, as a consequence, modifies the way the current motion is computed.
The state diagram is an abstraction layer of what the real state of the axis is, comparable to the image of the I/O points within a cyclic (PLC) program. A change of state is reflected immediately when issuing the corresponding motion command. (Note: the response time of ‘immediately’ is system dependent, coupled to the state of the axis, or an abstraction layer in the software).

The diagram is focused on a single axis. The multiple axis Function Blocks, MC_CamIn, MC_GearIn and MC_Phasing, can be looked at, from a state diagram point of view, as multiple single-axes all in specific states. For instance, the CAM-master can be in the state ‘Continuous Motion’. The corresponding slave is in the state ‘Synchronized Motion’. Connecting a slave axis to a master axis has no influence on the master axis.

The state ‘Disabled’ describes the initial state of the axis. In this state the movement of the axis is not influenced by the FBs. The axis feedback is operational.

If the MC_Power FB is called with Enable=TRUE while being in ‘Disabled’, this either leads to ‘Standstill’ if there is no error inside the axis, or to “ErrorStop” if an error exists.

Calling MC_Power with Enable=FALSE in any state, the axis goes to the state ‘Disabled’, either directly or via any other state. If a motion generating Function Block controls an axis, while the MC_Power FB with Enable=FALSE is called, the motion generating Function Block is aborted (CommandAborted). The intention of the “ErrorStop” state is that the axis goes to a stop, if possible. There are no further FBs accepted until a reset has been done from the ErrorStop state.

The transition Error refers to errors from the axis and axis control, and not from the Function Block instances. These axis errors may also be reflected in the output of the Function Blocks ‘FB instances errors’. Issuing MC_Home in any other state than StandStill will go to ErrorStop, even if MC_Home is issued from the state “Homing” itself. Function Blocks which are not listed in the State Diagram do not affect the state of the axis, meaning that whenever they are called the state does not change. They are: MC_ReadStatus; MC_ReadAxisError; MC_ReadParameter; MC_ReadBoolParameter; MC_WriteParameter; MC_WriteBoolParameter; MC_ReadActualPosition and MC_CamTableSelect. Calling the FB MC_Stop in state “StandStill” changes the state to “Stopping” and back to “Standstill” when “Execute = FALSE”. The state “Stopping” is kept as long as the input “Execute” is true. The “Done” output is set when the stop ramp is finished.
Fig 2.4 The State Diagram
### CHAPTER 3

**IMPLEMENTATION METHODS**

#### 3.1 FB INTERFACE

<table>
<thead>
<tr>
<th>Output exclusivity</th>
<th>The outputs ‘Busy’, ‘Done’, ‘Error’, and ‘CommandAborted’ are mutually exclusive: only one of them can be TRUE on one FB. If ‘Execute’ is TRUE, one of these outputs has to be TRUE. Only one of the outputs “Active”, ”Error”, “Done” and “CommandAborted” is set at the same time.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output status</td>
<td>The “Done”, “InGear”, “InSync”, “InVelocity”, “Error”, “ErrorID” and “CommandAborted” outputs are reset with the falling edge of “Execute”. However the falling edge of “Execute” does not stop or even influence the execution of the actual FB. It must be guaranteed that the corresponding outputs are set for at least one cycle if the situation occurs, even if execute was reset before the FB completed. If an instance of a FB receives a new execute before it finished (as a series of commands on the same instance), the FB won’t return any feedback, like ‘Done’ or ‘CommandAborted’, for the previous action.</td>
</tr>
<tr>
<td>Input parameters</td>
<td>The parameters are used with the rising edge of the execute input. To modify any parameter it is necessary to change the input parameter(s) and to trigger the motion again.</td>
</tr>
<tr>
<td>Missing input parameters</td>
<td>According to IEC 61131-3, if any parameter of a function block input is missing (“open”) then the value from the previous invocation of this instance will be used. In the first invocation the initial value is applied.</td>
</tr>
<tr>
<td>Position versus distance</td>
<td>“Position” is a value defined within a coordinate system. “Distance” is a relative measure related to technical units. “Distance” is the difference between two positions.</td>
</tr>
<tr>
<td>Sign Rules</td>
<td>The “Velocity”, “Acceleration”, “Deceleration” and “Jerk” are always</td>
</tr>
</tbody>
</table>
positive values. “Position” and “Distance” can be both positive and negative.

### Error Handling Behavior

All blocks have two outputs, which deal with errors that can occur while executing that Function Block.

These outputs are defined as follow:

<table>
<thead>
<tr>
<th>Error</th>
<th>Rising edge of “Error” informs that an error occurred during the execution of the Function Block.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ErrorID</td>
<td>Error number</td>
</tr>
</tbody>
</table>

“Done”, “InVelocity”, “InGear”, and “InSync” mean successful completion so these signals are logically exclusive to “Error”.

Types of errors:

- Function Blocks (e.g. parameters out of range, state machine violation attempted)
- Communication
- Drive

Instance errors do not always result in an axis error (bringing the axis to “StandStill”)

The error outputs of the relevant FB are reset with falling edge of Execute.

### FB Naming

In case of multiple libraries within one system (to support multiple drive / motion control systems), the FB naming may be changed to “MC_FBname_SupplierID”.

### Behavior of Done Output

The “Done” output (as well as “InGear”, “InSync”, ...) is set when the commanded action has been completed successfully.

With multiple Function Blocks working on the same axis in a sequence, the following applies:

when one movement on an axis is interrupted with another movement on the same axis without having reached the final goal, ”Done” of the first FB will not be set.
| Behavior of CommandAborted output | “CommandAborted” is set, when a commanded motion is interrupted by another motion command. The reset-behavior of “CommandAborted” is like that of “Done”. When “CommandAborted” occurs, the other output-signals such as “InVelocity” are reset. |
| Inputs exceeding application limits | If a FB is commanded with parameters which result in a violation of application limits, the instance of the FB generates an error. The consequences of this error for the axis are application specific and thus should be handled by the application program. |
| Behavior of Busy output | Every FB can have an output “Busy”, reflecting that the FB is not finished. “Busy” is SET at the rising edge of “Execute” and RESET when one of the outputs “Done”, “Aborted”, or “Error” is set. It is recommended that this FB should be kept in the active loop of the application program for at least as long as ‘Busy’ is true, because the outputs may still change. For one axis, several Function Blocks might be busy, but only one can be active at a time. Exceptions are “MC_SuperImposed” and “MC_Phasing”, where more than one FB related to one axis can be active. |
| Output ‘Active’ | The “Active” output is required on buffered Function Blocks. This output is set at the moment the function block takes control of the motion of the according axis. For un-buffered mode the outputs “Active” and “Busy” can have the same value. |
| Enable and Valid / Status | The “Enable” input is coupled to a “Valid” output. “Enable” is level sensitive, and “Valid” shows that a valid set of outputs is available at the FB. The “Valid” output is TRUE as long as a valid output value is available and the “Enable” input is TRUE. The relevant output value can be refreshed as long as the input “Enable” is TRUE. If there is a FB error, the output is not valid (“Valid” set to FALSE). When the error condition disappears, the values will reappear and “Valid” output will be set again. |
The behavior of “Execute”/”Done” style FBs is as follows:

Fig 3.1 Execute/Done style FBs [5]
3.2 Soft Motion

Everything from single to multi-axis movements to CAMs can be developed together with the logic application directly in the familiar IEC 61131-3 programming environment. A motion controller with CoDeSys SoftMotion implements the motion functionality in form of a toolkit integrated in the PLC Development System.

The Runtime System CoDeSys control SoftMotion has to be implemented on a motion controller in order to be able to program it with CoDeSys SoftMotion toolkit.

Platform Requirements:

- Has to be supported by CODESYS control (recommended min 32 bit with FPU).
- Needs sufficient performance for the calculation of the position, velocity and acceleration data.
- Needs hard real-time properties with low jitter values.

When working with CoDeSys SoftMotion controllers the user has many different options when it comes to implementing his motion tasks. For example, he can make use of the provided certified PLCopen motion POUs and appreciate the full functionality of the IEC 61131-3 development System.

CODESYS Motion + CNC is seamlessly integrated into the CODESYS Development System as a development kit and thus benefits from the available functions of the platform. Movement is processed in the controller within the context of CODESYS Control (IEC 61131-3 runtime system).

Configuration and commissioning of drives

- Adding the necessary bus system to the CODESYS project: support of CANopen, EtherCAT, and Sercos, as well as standard systems, such as stepper drives and drives with analog control; more upon request.
- Adding the required drives.
- The device name represents implicitly provided IEC data structure with abstracted data for each drive, allowing smooth exchange of drives and drive buses.
- Drive commissioning with integrated "Online Config Mode".
- Configuration of the bus-specific and drive-specific parameters via the object directory of the devices.
The SoftMotion technique involves the use of the CoDeSys platform in a systematic order based on the implementation of the functional blocks (FB). The various data from the HMI has to be communicated from the UI platform to the CoDeSys for which a common database is used. The different software used for the implementation of the overall working involves the following in the given order.

1) Database creation with specifications of the r/w and unique address of the different parameters used in the HMI.
2) Linking the UI to the respective external routines and programming the corresponding cases using the Visual studio.
3) Exporting the parameters created for HMI with unique addresses for communication to the CoDeSys Modbus.
4) Programming in CoDeSys for the execution of the application based on the user requirements.
5) Installing and configuring the required devices in the CoDeSys platform for the execution of the application. These devices can be the drivers and the motors that are based on application specific requirements.
6) Connecting to the device through the dedicated TCP/IP configuration settings.
7) Running the application.

The above steps imply the overall working of the SoftMotion technique implementation in the CoDeSys platform, and can be also seen using the algorithm given below:
The application is run on a virtual axis visualization prior to the real hardware, to make sure that there are no technical faults and to avoid any kind of risks that could damage the motors in case of mishandling or any wrong parameter input.

The HMI of the system is the major part of the overall application where the commands for the exchange of the data are sent from users for the execution of the program. Every action is related to the external routine and every changeable data in the UI is interpreted in terms of a dedicated parameter listed in the database.
This parameter, either in format of buttons or inputting values, when modified, sends the changed value to the processor to update them and perform the corresponding operations or calculations. Each button is application specific and performs an action based on the external routine linked with the Visual Studio.

Fig 3.4 Sample HMI for the Gasketing application
Every parameter in the database has a unique address which is addressed during the change of values during the execution of the application for the transfer of the information between the interface and the drivers. Once the parameters are created and listed, all the details are exported in the xml format. These exported files can be imported in the WinMachLite (WML) and also to the universal Modbus parameter declaration in CoDeSys in order to have a synchronized address match between the listed values and the parameters in the HMI.

The address generation and the Modbus list updates are followed by the device configuration of the driver in CoDeSys which is further logged into for the application test. The application is initially run on a virtual drive designed on the basis of visualization and then further implemented in the real drive once the system runs properly.

3.2.1 The virtual visualization of the drives:

We use the visualization templates which are available in the libraries and we connect them via the placeholder-concept with the function block instances.
Fig 3.6 Virtual visualization for observing the movement of the Axis

These visualizations are created in order to observe the virtual motion of the drive based on the inputs given to the interpolator. These can also be used to make the absolute and relative movements in the drives by providing the respective values of the velocity, acceleration, deceleration and direction in the corresponding blocks.

3.2.2 Motor configuration and drive interface with the communication bus

The motors used for the sample testing of the application involves the stepper motors whereas the real-time implementations in the 3D Cartesian robots involve the Servo motors. These motors have to be configured properly in order to have a precise movement. The tuning of the parameters of the motors plays a major role in the motion because the default movement is based on the number of counts of the motion whereas the user understands the SI units of the displacement. On studying the details of the motor the values of the counts are interpreted in terms of the linear movement and given as an input to the HMI.
The function, or task, of a servo can be described as follows. A command signal which is issued from the user's interface panel comes into the servo's "positioning controller". The positioning controller is the device which stores information about various jobs or tasks. It has been programmed to activate the motor/load, i.e. change speed/position.

The signal then passes into the servo control or "amplifier" section. The servo control takes this low power level signal and increases, or amplifies, the power up to appropriate levels to actually result in movement of the servo motor/load.

![Fig 3.7 Concept of a Servo system [3]](image)

These low power level signals must be amplified: Higher voltage levels are needed to rotate the servo motor at appropriate higher speeds and higher current levels are required to provide torque to move heavier loads.

This power is supplied to the servo control (amplifier) from the "power supply" which simply converts AC power into the required DC level. It also supplies any low level voltage required for operation of integrated circuits.

As power is applied onto the servo motor, the load begins to move, speed and position changes. As the load moves, so does some other "device" move. This other "device" is either a tachometer, resolver or
encoder (providing a signal which is "sent back" to the controller). This "feedback" signal is informing the positioning controller whether the motor is doing the proper job.

The positioning controller looks at this feedback signal and determines if the load is being moved properly by the servo motor; and, if not, then the controller makes appropriate corrections. For example, assume the command signal was to drive the load at 1000 rpm. For some reason it is actually rotating at 900 rpm. The feedback signal will inform the controller that the speed is 900 rpm. The controller then compares the command signal (desired speed) of 1000 rpm and the feedback signal (actual speed) of 900 rpm and notes an error. The controller then outputs a signal to apply more voltage onto the servo motor to increase speed until the feedback signal equals the command signal, i.e. there is no error.

Therefore, a servo involves several devices. It is a system of devices for controlling some item (load). The item (load) which is controlled (regulated) can be controlled in any manner, i.e. position, direction, speed. The speed or position is controlled in relation to a reference (command signal), as long as the proper feedback device (error detection device) is used.

![Diagram of servo system](image)

**Fig 3.8 Position control using a pulse signal [4]**

The feedback and command signals are compared, and the corrections made. Thus, the definition of a servo system is, that it consists of several devices which control or regulate speed/position of a load.

**Closed Loop Control**

A rotation detector (encoder) is mounted on the motor and feeds the rotation position/speed of the motor shaft back to the driver. The driver calculates the error of the pulse signal or analog voltage (position command/speed command) from the controller and the feedback signal (current position/speed) and controls the motor rotation so the error becomes zero. The closed loop control
method is achieved with a driver, motor and encoder, so the motor can carry out highly accurate positioning operations.

- An END signal is obtained that communicates the completion of the positioning operation.
- An alarm can be output if there is an abnormality such as an overload, making it possible to communicate equipment abnormalities

### 3.2.3 Position Control Diagram

![Position Control Diagram](image)

The Position control diagram has the feedback of the position and the speed from the encoder of the servo motor, which is given as an input to the speed and the current adjuster as a speed command and a current command. These values are compensated and given to the inverter for the change in the fed value and the original value to modulate the current accordingly to make the motor move in the right speed at the right position.
3.3 The SoftMotion Drive Interface

The SoftMotion Drive Interface is a standardized interface, which allows to include the abstracted image of a drive hardware in the IEC program, and to configure and address it there. It takes care of update and transfer of the motion data which are necessary for controlling the drive hardware. This not only allows easy changing of drives and reuse of IEC programs, but also saves the difficulties and inconveniences of connecting the drives.

The Drive Interface uses the following components:

- **The CoDeSys PLC Configuration**: Here – basing on a corresponding configuration file - the structure of the drives, which should be controlled, must be mapped by the programmer and the appropriate parameters have to be set. This structure then will be made accessible for the application with the aid of the Drive Interface libraries by implicitly created and assigned (system) variables.

- **The internal library Drive_Basic.lib**: provides IEC data structures and global variables, which will represent the drives, axis groups and bus interfaces which have been configured in the PLC Configurator.

- **The driver, i.e. the hardware and bus system specific library Drive.lib (e.g. SercosDrive.lib)**, which has to be provided by the drive manufacturer offers special functions for the data exchange between the structures and the hardware.

![Fig 3.10 Data exchange between structures and the hardware. [5]](image-url)
3.4 Error Handlings

All access to the drive/motion control is via Function Blocks. Internally these Function Blocks provide basic error checking on the input data. Exactly how this is done is implementation dependent. For instance, if MaxVelocity is set to 6000, and the Velocity input to a FB is set to 10,000, a basic error report is generated. In the case where an intelligent drive is coupled via a network to the system, the MaxVelocity parameter is probably stored on the drive. The FB has to take care that it handles the error generated by the drive internally. With another implementation, the MaxVelocity value could be stored locally. In this case the FB will generate the error locally.

Both centralized and decentralized error handling methods are possible when using the motion control Function Blocks.

Centralized error handling is used to simplify programming of the Function Block. Error-reaction is the same independent of the instance in which the error has occurred.

Fig 3.11 Function blocks with centralized error handling.
Decentralized error handling gives the possibility of different reactions depending on the Function Block in which an error occurred.

Fig 3.12 Function blocks with decentralized error handling.
CHAPTER 4

ANALYSIS OF SOFTPLC AND SOFTMOTION LIBRARY

4.1 IMPLEMENTATION OF GASKETING USING THE SOFTMOTION TECHNIQUE

OVERVIEW

The CNC-Editor in CoDeSys allows to program multi-dimensional motions graphically and textually in parallel, following the CNC language DIN66025.

Basically up to 9-dimensional motions can be realized, whereby only two dimensions are not interpolated linearly. Thus in two dimensions lines, circles, circular arcs, parables and splines can be programmed; the other directions merely get interpolated linearly.

For each programmed path CoDeSys automatically creates a global data structure (CNC Data), which can be used by the IEC program.

This can be done in different ways:

1. The CNC program is stored as an array of G-Code-Words and will be decoded during runtime of the PLC program with the aid of a decoder module. Thus for the particular path objects GEOINFO structure objects will be available. path-preprocessing modules (see SM_CNC.lib, e.g. Tool Radius Correction), afterwards interpolated, transformed and returned to the Drive Interface for the communication with the hardware. (see command 'Create program variable on compile').

2. The CNC program is written as a list (OUTQUEUE-Structure) of GEOINFO structure objects to a data structure and thus can directly be fed to the interpolator. In comparison to a) by this method you can avoid calling the Decoder and the Path Preprocessing Modules. But in exchange you do not have the possibility to change the program during runtime. (see command 'Create OutQueue file on compile').
3. The CNC program is written as described in a resp. b to the file system of the controller and is read and executed at runtime step by step. This method particularly is suitable for big programs, which cannot be kept completely in the memory.

The gasketing application involves a trajectory over which the motors have to move precisely. The movement is given by the Gcodes which is given by the user directly in the CNC window of CoDeSys as shown in fig 4.1. This CNC file is given as an executable command to the SMC interpolator function block which fetches the values of the motion and runs the drivers correspondingly.

![G-code generation for the required motion](image)

Every path in the CNC window is designed by the user as per the application requirements. Once these Gcodes are written, we have a preview of the motion that needs to be followed in the system application. This helps the user to understand the precise movement preview of the path to be followed.

This CNC code file is given a unique name which is given as an input to the SMC interpolator (Fig 4.2) that translates the various parameters for the movements in the various directions based on the Gcode values given by the user.
Fig 4.2 Assigning the G code file to the SMC interpolator block and configuring the motion.

The data passed to the SMC interpolator are interpolated to the 3D trafo gantry to fetch the values of the three axis and are given as an input to the SMC_controlAxisByPos FB which helps to control the axis by inputting the required values to the drives.

4.2 SMC_Interpolator

The SMC_Interpolator function block is used to convert a continuous path which is described by SMC_GEOINFO-objects, into discrete path position points, thereby regarding a defined velocity profile and time pattern. These position points typically afterwards will be transformed by the IECprogram (e.g. to drive-axis-positions) and sent to the drives.
4.2.1 Inputs of the function block:

**bExecute**: BOOL  The function block will do a reset and start the path correction (rounding path) as soon as a rising edge is detected at this input.

**poqDataIn**: POINTER TO SMC_OUTQUEUE This variable points to the SMC_OUTQUEUE-structure object, which contains the SMC_GEOINFO-objects of the unsmoothed path; typically it points to the output .DataOut of the preceding module SMC_CheckVelocities.

**bSlow_Stop**: BOOL  If this variable is set to FALSE (default), the path will be passed non-stop. With TRUE the SMC_Interpolator will be caused to reduce the velocity to 0 – according to the defined velocity profile (byVelMode, see below) and the maximum delay of the current GEOINFO-object (dDecel, see below) – and to wait until bSlow_Stop will be reset to FALSE.

**bEmergency_Stop**: BOOL  This input per default is FALSE. As soon as it gets TRUE, the SMC_Interpolator will cause an immediate stop, this means that the position will be retained. Thus the velocity will be set to 0 immediately.

**bWaitAtNextStop**: BOOL  As long as this variable is FALSE (default), the path is passed non-stop. With TRUE the SMC_Interpolator will be caused to retain the position at the next regular stop – this means at position points where the velocity is 0, typically at path angles – and to pause until bWaitAtNextStop will be reset to FALSE.

**dOverride**: LREAL  This variable can be used to handle the override. Valid values are higher than 0.01. dOverride is multiplied with the scheduled velocity of the particular objects and thus allows to increase resp. reduce the scheduled velocity in online mode. For example, dOverride=1 (default) effects that the programmed scheduled velocities will be executed, while an dOverride=2 would double them. Please regard: The override can be modified at any time, but the modification will only be applied, if currently no acceleration or deceleration is in progress.

**iVelMode**: SMC_INT_VELMODE  This input defines the velocity profile. The value "TRAPEZOID" (default) effects a velocity profile which has a trapezoid form, "SIGMOID" one which has a S-form:
Fig 4.3 Sigmoidal and Trapezoidal Velocity Profile.

In the examples shown above the maximum acceleration (Accel) is lower than the maximum deceleration (Decel). This causes the different slope values of the velocity curve at acceleration and deceleration. The advantage of a sigmoid velocity profile is that the associated acceleration – in contrast to the trapezoid – is continuous and thereby brings a relief especially for heavy machines. This must be paid by a slightly increased calculating time.

As the sigmoid velocity profile (blue) is designed in a way that changing to the trapezoid profile (red) does not result in a change of the time needed for the passing the complete path, the restrained increase
of the acceleration at the beginning and the end must be compensated by a higher acceleration in the midway. Thereby you must regard, that the maximum acceleration resp. deceleration, which is programmed in the SMC_GEOINFO-objects, will be exceeded in the maximum by the factor $\pi/2$:

Any online-change will not be applied – like described for dOverride – until a currently running acceleration or deceleration has been terminated.

In order to interpolate also the additional axes in the sigmoid form (blue, see drawing below) instead of linearly (red), the corresponding bits must be set in the variable wSProfile of piStartPos of the current object. This will effect that the additional axis does not get interpolated linearly concerning the path length in the X-, Y-, Z-space, but in a polynomial dependency on this path length, thus resulting in a sigmoid profile for the axis position, which has a velocity and acceleration of 0 at the beginning and at the end of a path segment.

![Diagram]

dwIpoTime: DWORD This variable, which must be set for each call, contains the cycle time in $\mu$sec.

dLastWayPos: LREAL This input allows the user to measure the stretch of the path which is racked out by the Interpolator. Output dWayPos is the sum of dLastWayPos and the distance covered within the current cycle. If dLastWayPos=0, dWayPos shows the length of the current path segment. If dLastWayPos is set equal to output dWayPos, dWayPos always will be incremented by the current path segment and you get the total length of the already covered path. In doing so dLastWayPos at any time can be (re)set to 0 or a different value.

bSingleStep: BOOL This input effects that the interpolator will stop at the end of each path object (also at transitions with identic tangent) and wait until this input is reset to FALSE. If bSingleStep is set TRUE during the move, the interpolator will stop at the end of that object, which it can reach without exceeding the scheduled deceleration value.

bAbort: BOOL This input aborts the processing of an outline.
4.2.2 Outputs of the function block:

**bDone: BOOL**
This variable will be set to TRUE as soon as the input data (poqDataIn) have been processed completely. Hereafter the function block will not perform any further actions until a reset will be done. If input bExecute is FALSE, bDone will be reset to FALSE.

**bError: BOOL**
In case of an error this input gets TRUE.

**wErrorID: SMC_ERROR (INT)**
This enumeration variable may describe an error which has occurred during the interpolation run. After an error has occurred, the processing gets stopped until a Reset will be done.

Per each call the SMC_Interpolator – considering the predefined parameters, the velocity history and the last position – will calculate and put out the next point. As soon as the processing of currently first GEOINFO-object has been finished, it will be removed from the poqDataIn-SMC_OUTQUEUEstructure.

**piSetPosition: SMC_POSINFO**
This variable contains the target position which has been calculated according to the predefines. Set_Position is a SMC_POSINFO-structure and not only contains the Cartesian coordinates of the target point on the path, but also the position of the additional axes.

**iStatus: INT_STATUS (INT)**
This enumeration variable shows the current status of the function block. Possible states:

<table>
<thead>
<tr>
<th>IPO_UNKNOWN</th>
<th>0</th>
<th>Internal state. This state may not occur after a complete pass of the SMC_Interpolator.</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPO_INIT</td>
<td>1</td>
<td>Module is in initialization state; DataIn currently is not full and also not yet has been full. IPO_ACCEL 2 Module currently is accelerating.</td>
</tr>
<tr>
<td>IPO_ACCEL</td>
<td>2</td>
<td>Module currently is accelerating</td>
</tr>
<tr>
<td>IPO_CONSTANT</td>
<td>3</td>
<td>Module currently is moving with constant velocity.</td>
</tr>
<tr>
<td>IPO_DECEL</td>
<td>4</td>
<td>Module currently is decelerating</td>
</tr>
<tr>
<td>IPO_FINISHED</td>
<td>5</td>
<td>Processing the GEOINFO list is terminated. Any further GEOINFO objects which arrive subsequently in DataIn will not be processed.</td>
</tr>
</tbody>
</table>
| IPO_WAIT    | 6 | Module is waiting, because one of the following situations has occurred.  
Emergency_Stop = TRUE.  
Slow_Stop = TRUE and Vel = 0.  
Wait_At_Next_Stop = TRUE and Vel = 0. |

**Table 4.1 Interpolator States**

**bWorking: BOOL**

This output gets TRUE, as soon as the processing of the list has been started but is not yet finished (IPO_ACCEL or IPO_CONSTANT or IPO_DECEL or IPO_WAIT). Otherwise bWorking is FALSE.

**iActObjectSourceNo: INT**

Here you find the value of SourceLine_Nr of the currently passed GEOINFO-object of the DataInqueue. If the SMC_Interpolator does not work (any longer) (Working = FALSE), the value is "−1".

**dVel: LREAL**

This variable contains the current velocity which results if an object is moving from the preceeding position to Set_Position within the given time Ipo_Time.

**vecActTangent: SMC_VECTOR3D**

This structure contains the direction of the path valid for the position Set_Position. If Vel = 0, vecAct_Tangent is filled up with zeros.

**iLastSwitch: INT** This output shows the number of the last passed switch. Regard: If several switches have been passed within one cycle, only the last one will be shown.

**dwSwitches: DWORD** This DWORD describes the current switch status of all switches 1 – 32. Bit0 of the DWORD represents switch1, Bit31 represents switch321, Bit31 for auxiliary mark 32. Thereby, in contrast to iLastSwitch, it can be avoided that any switch is not regarded.
**dWayPos**: LREAL For a description see above: input dLastWayPos.

At each call the SMC_Interpolator will calculate and provide - regarding the given parameters, the velocity history and the last path position - the subsequent path position point. If the first GEOINFO object has been processed, it will be removed from the poqDataIn-SMC_OUTQUEUE structure.

### 4.3 ControlAxis function blocks

These modules can be used to control a drive by direct setting of the desired values:

#### 4.3.1 SMC_ControlAxisByPos

This function block writes target positions n to a drive structure and checks the structure for jumps.

<table>
<thead>
<tr>
<th>SMC_CONTROLAXISBYPOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>iStatus : SMC_INT_STATUS bCommandAborted : BOOL</td>
</tr>
<tr>
<td>bEnable : BOOL bError : BOOL</td>
</tr>
<tr>
<td>bAvoidGaps : BOOL iErrorID : SMC_ERROR</td>
</tr>
<tr>
<td>fSetPosition : LREAL bStopIpo : BOOL</td>
</tr>
<tr>
<td>fGapVelocity : LREAL Axis : AXIS_REF (VAR_IN_OUT)</td>
</tr>
<tr>
<td>fGapAcceleration : LREAL</td>
</tr>
<tr>
<td>fGapDeceleration : LREAL</td>
</tr>
<tr>
<td>Axis : AXIS_REF (VAR_IN_OUT)</td>
</tr>
</tbody>
</table>

#### 4.3.1.1 Inputs of the function block:

**iStatus**: SMC_INT_STATUS

State of the Interpolation module. Gets connected with the homonymous output of SMC_Interpolator.

**bEnable**: BOOL Controls the axis as long as is TRUE.

**bAvoidGaps**: BOOL TRUE:

The module watches position and velocity. If the velocity exceeds the limit fSWMaxVelocity, which is stored in the axis (configured in the drive dialog in „Maximum values“), then the module will set output bStopIpo and move the axis according to the parameters fGapVelocity, fGapAcceleration and fGapDeceleration to this position and then will delete output bStopIpo.

**fSetPosition**: LREAL

Target position of the axis. Typically this is an output of the Transformation module.
fGapVelocity, fGapAcceleration, fGapDeceleration: LREAL

Move parameters for bridging a jump.

**4.3.1.2 Outputs of the function block:**

- **bCommandAborted : BOOL (Default: FALSE) TRUE:**
  The module has been aborted by another one.

- **bError : BOOL (Default: FALSE)**
  TRUE: An error has occurred in the module.

- **iErrorID : SMC_Error (INT )**
  Error number

- **bStopIpo : BOOL (Default: FALSE)**
  TRUE: the module has detected a jump in velocity or position and is just adjusting to the new position. For this reason, this output should be connected to the EmergencyStop-input of the SMCInterpolator, so that the Interpolator will wait until the axis is positioned correctly.

The Gcodes created in the CNC editor window, once they are passed through the SMC interpolator Function block and the control axis by position function blocks are executed by the interpolator; where the axis enabling and the command execution takes place. These tasks are executed by creating a program in the CoDeSys PLC application structure.

**4.4 HOMING**

One important feature of the similar mechanical applications like gasketing involves homing. It involves the configuration of the current position as the start position of the driver to move. This is usually preferred to be done in the manual mode of operation in order to give the liberty to the user to place the starting position of the drive to the desired position. This start position is where the driver starts to move from and finishes the task once the path has been completed. The homing application can be implemented either by using the FB defined in the SoftMotion window or by using the manual calculations that can be achieved by acquiring the positions of the drives.

The Functional Block representation of the SMC_HOMING can be shown as below:
This function block can execute the reference move of an axis. As an ON-switch a boolean value is used, typically a hardware input.

After the module has been started with a rising edge in \textit{bExecute}, it moves the axis at a velocity \textit{fVelocityFast} in a direction defined by \textit{nDirection}, until the \textit{bEndSwitch} = \text{FALSE}, i.e. the reference switch, will be closed. Then the axis will be slowed down and driven in the opposite direction according to \textit{fVelocitySlow}. The reference position will be set and the drive will be stopped at exactly that point where the reference switch opens (\textit{bEndSwitch} = \text{TRUE}).

\subsection*{4.4.1 Inputs of the module:}

\textbf{bExecute : BOOL} (Default: FALSE)

At a rising edge the reference motion of the drive will be started.

\textbf{fHomePosition : REAL}

Absolute position on the reference position [u].

\textbf{fVelocitySlow, fVelocityFast : REAL}

Target velocity for phase 1 and 2 in [u/s].

\textbf{fAcceleration, fDeceleration : REAL}

Target acceleration and deceleration in [u/s^2].
nDirection : MC_Direction (Default: negative)

Direction of the reference motion: permissible values: positive/negative.

bEndSwitch : BOOL (Default: TRUE)

Reference switch: TRUE (open), FALSE (closed).

fSignalDelay : REAL (Default: 0.0)

Transmission time of the reference switch in s. If a time >0 is set, the module will not use the position at which the bEndSwitch has got TRUE as a reference position, but that position which the axis had fSignalDelay seconds before.

nHomingMode : SMC_HOMING_MODE (Default: FAST_BSLOW_S_STOP)

**FAST_BSLOW_S_STOP:**

The drive will be moved to the given direction at velocity $f_{VelocityFast}$ (FAST) until the input $bEndSwitch$ gets FALSE, then will be stopped and moved to the opposite direction at velocity $f_{VelocitySlow}$ (BSLOW) until $bEndSwitch$ gets TRUE again. At this position the reference point will be set (S) and it will be stopped (STOP).

**FAST_BSLOW_STOP_S:**

In contrast to FAST_BSLOW_S_STOP after the free move first a stop is done and afterwards the reference point is set.

**FAST_BSLOW_I_S_STOP:**

In contrast to FAST_BSLOW_S_STOP after the first free move an index impulse ($b_{IndexOccured}$=TRUE) and its position $f_{IndexPosition}$, set as reference point, will be awaited. Not until then it will be stopped.

**FAST_BSLOW_S_STOP/ FAST_BSLOW_STOP_S / FAST_BSLOW_I_S_STOP:**

These modes work exactly like those described above except that there will not be turned reverse when having reached the reference switch but will be moved on Regard that in this modes input $b_{IgnoreHWLimits}$ for safety reasons must be FALSE.

bReturnToZero: BOOL (Default: FALSE)
If this flag is set, the module will set the position on the zero point after having terminated the procedure which is defined by \textit{nHomingMode}.

\textbf{bIndexOccured: BOOL} (Default: FALSE)

Only for \textit{nHomingMode FAST_BSLOW_I_S_STOP}: Indicates whether the index pulse has occurred.

\textbf{fIndexPosition: REAL} (Default: 0.0)

Only for \textit{nHomingMode FAST_BSLOW_I_S_STOP}: Latched position of the index pulse. If this entry is TRUE, the hardware control of the end switches will be intermitted. Choose this option if you use the same physical switch as hardware-end and reference switch.

\textbf{bIgnoreHWLimit: BOOL} (Default: FALSE)

As long as this input is TRUE, the hardware control of the end switches will not be done. Use this option, if you are using the same physical switch for the hardware end switch and the reference switch.

\subsection*{4.4.2 Outputs of the module:}

\textbf{bDone : BOOL} (Default: FALSE)

If TRUE, the drive is referenced and in standstill.

\textbf{bCommandAborted : BOOL} (Default: FALSE)

If TRUE, the command has been aborted by another one.

\textbf{bError : BOOL} (Default: FALSE)

TRUE indicates a function block error.

\textbf{nErrorID : SMC_Error} Error number.

\section*{4.5 Drive Interface: Create PLC Configuration for Drives}

This describes how to map a given physical drive structure to the IEC program in the CoDeSys programming system. By doing this configuration the IEC program will get access to data structures, which can be used by the SoftMotion modules to create the desired movements.

- First open the PLC Configuration (Resources tab) and select 'Extras' 'Standard configuration'.
• For this example it is assumed, that a Sercos field bus is used and e.g. an ISA-Bus-Card from Automata. Hence use the command 'Append BusInterface' in the context menu (right mousebutton).

• For this bus interface now the appropriate module parameters (Modulparameter) have to be set. First define the "InterfaceType". In our example no hardware is available and we set a kind of simulation mode by selecting "Dummy".

• Depending on the chosen InterfaceType the further parameters have to be defined: E.g. for "Sercos (Automata)" you have to set the interrupt number in wParam1, the hardware type in wParam2 and the base address of the card in dwParam1. Thus - the entry 'BusInterface' in the configuration tree must be selected (dotted frame) – click in the configuration tree window with the right mouse-button and select command 'Append AxisGroup'. For the AxisGroup now define the Modulparameters. First enter the name of the Task (sTask), which will control the drives (e.g. "BusTask"), in the line below enter the cycle time of the task in µsec, e.g. "3000". (See below for the creation of this task.) The other AxisGroup parameters depend on which BusInterface has been chosen. In our example wParam1 defines the baud rate in MBit (e.g. enter "2") and wParam2 defines the intensity of the LED.

• Create a task for the drive control with the following attributes in the Task configuration:
• Now the drives have to be inserted. We assume that there are four drives, three linear drives serving a X-, Y-, Z- portal-system, and one drive turning the tool around the Z-axis. Insert each drive by the command "Append Drive", which is available in the context menu when the entry 'AxisGroup' is selected (dotted line) in the configuration tree. The names of the drives can be modified. For this purpose click on the entry with the right mouse-button to open an edit field.

For our example we choose the following names:

• Now we are going to parameterize the drives: For this purpose we open the ModulparameterDialog:

• First define the ID "wDriveId" according to the drive configuration. In our example: "1", "2", "3" and "4". In the following it will be described how the settings in tab „Module parameters“ have to be done; much easier and self-explanatory that also can be done via the dialogs.

• The portal-drives each can be moved between -50cm and +50cm. We configure their weighting translational. We use an increment of 10-7 m resp. 10-7m/s for all position and velocity data.
Thus all position and velocity data basically have to be evaluated with the unit "10-7m" resp. "10-7m/s". If we want to change this to a unit of "mm/sec", we have to enter "10000" for the parameter dwRatioTechUnitsDenom "10000" and "1" for iRatioTechUnitsNum. Due to the fact that it is a linear drive, fPositionPeriod has no meaning. But you still have to specify the data, which will be sent and received cyclically. For example: If we refer to master telegram, it is sufficient to choose "POS, VEL -> POS, VEL" in the scroll list of wControlType. This will cause that cyclically the position and velocity target values will be sent and the current position and velocity values will be received.

In order to have an additional control of any exceeding of the valid range (-50cm = -5000mm, 50 cm = 5000mm) (the application should be programmed in a way, that this is not possible at all), we activate a control function by setting SWLimitEnable = TRUE, SWLimitNegative = -5000 and SWLimitPositive = 5000.

- We assume a rotation of 65536 increments for the used rotatory drive can be rotated arbitrarily. Thus we define – in order to get an internal unit of angle degrees - "65536" for dwRatioTechUnitsDenom and "360" for iRatioTechUnitsNum. This drive for example might be designed for turning a screwtop on a bottle. Therefore we want to send cyclically the position and the torque values, because we later want to be able to switch - by a change of the operation mode - from a behaviour which is controlled by the position to a behaviour which is controlled by the torque. To do this settings select "CONFIGURABLE" for wControlType and "fSetPosition" resp. "fSetTorque" for wCyclicDataS1 resp. wCyclicDataS2. In order to get returned the current position set the option "fActPosition" for wCyclicDataR1.

- In order to get a program ready for an error-free compilation, a program call has to be appended to the task "BusTask". For example create a program "Ipo", which later will do the motion control, and call it by "BusTask".

Now compile the program, no errors should occur, and load it to the controller and start it. CoDeSys will automatically create the following variables and structures:

- In the Global Variables folder "Drive Configuration Data" you will find three instances of the modules "SercosDriveExecute_Start", "SercosDriveExecute_End" and "SercosDriveInit" (elements of the Sercos-library) with the names "AxisGroupStartCycle", "AxisGroupEndCycle" and "AxisGroupInit", which are responsible for the communication with the drives.
In the Global Variables folder "Drive_Globale_Variablen" of the library "Drive_Basic.lib" there is a structure variable g_DRVESTRUCT, which contains all entries of the PLC Configuration, i.e. all BusInterfaces, AxisGroups and Drives.

Besides that for each drive globally a structure variable has been created, e.g. "X_Drive", which can be monitored e.g. in the Watch- and Receipt Manager. This structure can be accessed by the SoftMotion modules and the DriveInterface will keep it up-to-date.

4.6 Direct Creation of the OutQueue

4.6.1 Creation of the NC program in the CNC-Editor:

We create an example program, which is running between x out of [0,100] and y out of [0,100]. Additionally we define the velocities and accelerations for the path and set two witch points on the path.
4.7 Dynamic SoftMotion Programming

One of the advantages of SoftMotion is, that the programmer and the user not only can influence the processing of a path but also can create and modulate this path while the program is running. In order to reach this the programmer just has to replace the Decoder module by an self-designed path generator. Nevertheless the path preprocessing and especially the Interpolator can be used further on as usual. In order to replace the Decoder module, the OUTQUEUE-structure object must be created in another way. It must also be filled with GEOINFO-objects, which represent the desired path, and it must be passed on to the appropriate subsequent module (e.g. Interpolator).

Preparing steps:

- In the declaration part an OUTQUEUE-, a GEOINFO-structure object and a buffer of desired size must be defined:

  QUEUE: SMC_OUTQUEUE:=(nSize:=10000);
  BUF: ARRAY[0..9999] OF BYTE;
  GEO: SMC_GEOINFO:=(dT1:=0, dT2:=1, dToolRadius:=0, dVel:=100, dVel_End:=100, Accel:=200, dDecel:=500, iObj_Nr:=0);

- In an Init-Step in the implementation part of the program the OUTQUEUE-structure must get initialized:

  QUEUE.nSize := 10000;
  QUEUE.pbyBuffer := ADR(BUF[1]);

The steps also involves the parameter declaration and the conversion of the Gcode declaration in the CNC window to the corresponding Geoinfo group to accumulate the parameters in a geoinfo array for the corresponding execution of the path.

Declaration of the Global variable for the generation of the Geoinfo datas from the Gcodes created in the CNC Editor:

```
VAR_GLOBAL
  //MAXPOSITIONINPATH: INT:=100;
  AutomataPath: SM3_CNC.SMC_OUTQUEUE;
  AutomataGeoInfo: ARRAY[1..100] OF SM3_CNC.SMC_GEOINFO;
  AutomataPathGCode: SM3_CNC.SMC_OUTQUEUE;
  AutomataGeoInfoGCode: ARRAY[1..100] OF SM3_CNC.SMC_GEOINFO;
END_VAR
```
This is followed by the programming format for the conversion of the datas to the required parameters.

### 4.8 Program code:

**For conversion of the Gcode to the GeoInfo and storing the data in the SMC_OUTQUEUE**

```plaintext
IF CMD_GasketingPathGenerated THEN
    CMD_GasketingPathGenerated: =FALSE;

// Path Creation
AutomataPath.wOUTQUEUEStructID: =16#9878;
AutomataPath.strProgramName: =WSTRING_TO_STRING("CANNON-Automata");
AutomataPath.pbyBuffer: =ADR (AutomataGeoInfo [1]);
AutomataPath.byGenerator: =255;
GCodeGeoInfo.iObjNo: =0;

count: =0;

hFile: =SysFileStream23.SysFileStreamFOpen(FileName: =WSTRING_TO_STRING("HARDISK//GASKETING.gcode"), Mode:='r');
WHILE (SysFileStream23.SysFileStreamFEOF(File: =hFile) = 0) DO
    GCODE: =SysFileStream23.SysFileStreamFGetC(File: =hFile);
    GCODEW: =DINT_TO_WORD(GCODE);
    GCODE1: =WORD_AS_STRING (GCODEW, FALSE);
    IF (GCODE1 = 'N') THEN
        SysFileStream23.SysFileStreamFScanf_Real (File: =hFile, szFormat: =WSTRING_TO_STRING("%d "), pfArg: =ADR(NData));
        GCodeGeoInfo.iObjNo: =GCodeGeoInfo.iObjNo+1;
        GCodeGeoInfo.iSourceLineNo[GCodeGeoInfo.iObjNo]: =NData;
        END_IF
    IF (GCODE1 = 'G') THEN
        GCODE: =SysFileStream23.SysFileStreamFGetC(File: =hFile);
        GCODEW: =DINT_TO_WORD(GCODE);
        GCODE1: =WORD_AS_STRING (GCODEW, FALSE);
        Gdata: =STRING_TO_DINT(GCODE1);
        CASE GData OF
            0:
                GcodeGeoInfo.iMoveTyp[GCodeGeoInfo.iObjNo]: =SM3_CNC.LINPOS;
                GCodeGeoInfo.dP1[GCodeGeoInfo.iObjNo]: =0;
                GCodeGeoInfo.dP2[GCodeGeoInfo.iObjNo]: =0;
                GCodeGeoInfo.dP3[GCodeGeoInfo.iObjNo]: =0;
                GCodeGeoInfo.dT1[GCodeGeoInfo.iObjNo]: =0;
                GCodeGeoInfo.dT2[GCodeGeoInfo.iObjNo]: =0;
            1:
                GCodeGeoInfo.iMoveTyp[GCodeGeoInfo.iObjNo]: =SM3_CNC.LIN;
                GCodeGeoInfo.dP1[GCodeGeoInfo.iObjNo]: =0;
                GCodeGeoInfo.dP2[GCodeGeoInfo.iObjNo]: =0;
                GCodeGeoInfo.dP3[GCodeGeoInfo.iObjNo]: =0;
                GCodeGeoInfo.dT1[GCodeGeoInfo.iObjNo]: =0;
                GCodeGeoInfo.dT2[GCodeGeoInfo.iObjNo]: =0;
            2:
                GCodeGeoInfo.iMoveTyp[GCodeGeoInfo.iObjNo]: =SM3_CNC.CLW;
            3:
```

// The “CMD_GasketingPathGenerated” is the parameter checked from the HMI, if its activated or deactivated.
GCodeGeoInfo.iMoveTyp[GCodeGeoInfo.iObjNo] = SM3_CNC.CCLW;

END_CASE
END_IF

IF (GCODE1 = 'X') THEN
  SysFileStream23.SysFileStreamFScanf_Real (File: = hFile, zFormat: = WSTRING_TO_STRING ("%f "), pfArg: = ADR(XData));
  GCodeGeoInfo.piDestPos[GCodeGeoInfo.iObjNo].dX: = XData;
END_IF
IF (GCODE1 = 'Y') THEN
  SysFileStream23.SysFileStreamFScanf_Real (File: = hFile, szFormat: = WSTRING_TO_STRING ("%f "), pfArg: = ADR(YData));
  GCodeGeoInfo.piDestPos[GCodeGeoInfo.iObjNo].dY: = YData;
END_IF
IF (GCODE1 = 'Z') THEN
  SysFileStream23.SysFileStreamFScanf_Real (File: = hFile, szFormat: = WSTRING_TO_STRING ("%f "), pfArg: = ADR(ZData));
  GCodeGeoInfo.piDestPos[GCodeGeoInfo.iObjNo].dZ:= ZData;
END_IF
IF (GCODE1 = 'R') THEN
  SysFileStream23.SysFileStreamFScanf_Real (File: = hFile, szFormat: = WSTRING_TO_STRING ("%f "), pfArg: = ADR(RData));
  GCodeGeoInfo.dP3[GCodeGeoInfo.iObjNo]: = RData;
END_IF
IF (GCODE1 = 'F') THEN
  SysFileStream23.SysFileStreamFScanf_Real (File: = hFile, szFormat: = WSTRING_TO_STRING ("%f "), pfArg: = ADR(FData));
  GCodeGeoInfo.dVel[GCodeGeoInfo.iObjNo]: = FData;
  GCodeGeoInfo.dVelEndStore[GCodeGeoInfo.iObjNo]: = FData;
  GCodeGeoInfo.dAccel[GCodeGeoInfo.iObjNo]: = FData/2;
  GCodeGeoInfo.dDecel[GCodeGeoInfo.iObjNo]: = FData/2;
END_IF
END_WHILE
SysFileStream23.SysFileStreamFClose(File: = hFile);
FOR i: = 1 TO GcodeGeoInfo.iObjNo-1 BY 1 DO
  AutomataGeoInfo[i].dVelEnd: = GcodeGeoInfo.dVel[i];
END_FOR
FOR i: = 1 TO GcodeGeoInfo.iObjNo BY 1 DO
  AutomataGeoInfo[i].iObjNo: = i;
  AutomataGeoInfo[i].iSourceLine_No: = GCodeGeoInfo.iSourceLineNo[i];
  AutomataGeoInfo[i].iMoveType: = GCodeGeoInfo.iMoveTyp[i];
  AutomataGeoInfo[i].piDestPos: = GCodeGeoInfo.piDestPos[i];
  IF i > 1 THEN
    AutomataGeoInfo[i].piStartPos: = GCodeGeoInfo.piDestPos[i-1];
  END_IF
  AutomataGeoInfo[i].dVel: = GCodeGeoInfo.dVel[i];
  AutomataGeoInfo[i].dVelEndStore: = GCodeGeoInfo.dVelEndStore[i];
  AutomataGeoInfo[i].dAccel: = GCodeGeoInfo.dAccel[i];
  AutomataGeoInfo[i].dDecel: = GCodeGeoInfo.dDecel[i];
  AutomataGeoInfo[i].dP3: = GCodeGeoInfo.dP3[i];
  IF i> 1 THEN
    GcodeGeoInfo.piStartPos[i].dX: = GCodeGeoInfo.piDestPos[i-1].dX;
    GcodeGeoInfo.piStartPos[i].dY: = GCodeGeoInfo.piDestPos[i-1].dY;
    GcodeGeoInfo.piStartPos[i].dZ: = GCodeGeoInfo.piDestPos[i-1].dZ;
  END_IF
END_IF
END_FOR
FOR i := 1 TO Gcodegeoinfo.iObjNo BY 1 DO
IF(Gcodegeoinfo.iMoveTyp[i] = SM3_CNC.CLW OR Gcodegeoinfo.iMoveTyp[i] = SM3_CNC.CCLW) THEN
    a := EXP((GcodeGeoinfo.piStartPos[i]. dX - GcodeGeoInfo.piDestPos[i]. dX),2);
    b := EXP((GcodeGeoinfo.piStartPos[i]. dY - GcodeGeoInfo.piDestPos[i]. dY),2);
    length := (SQRT(a+b));
    x_mid := (GcodeGeoInfo.piStartPos[i]. dX+GcodeGeoInfo.piDestPos[i]. dX)/2;
    y_mid := (GcodeGeoInfo.piStartPos[i]. dY+GcodeGeoInfo.piDestPos[i]. dY)/2;
    IF(GcodeGeoinfo.iMoveTyp[i] = SM3_CNC.CLW) THEN
        x_center := x_mid - SQRT(EXPT(Gcodegeoinfo.dP3[i],2) - EXPT((length/2),2)) * ((GcodeGeoInfo.piStartPos[i]. dY - GcodeGeoInfo.piDestPos[i]. dY)/length);
        y_center := y_mid - SQRT(EXPT(Gcodegeoinfo.dP3[i],2) - EXPT((length/2),2)) * ((GcodeGeoInfo.piDestPos[i]. dX - GcodeGeoInfo.piStartPos[i]. dX)/length);
    ELSEIF(Gcodegeoinfo.iMoveTyp[i] = SM3_CNC.CCLW) THEN
        x_center := x_mid + SQRT(EXPT(Gcodegeoinfo.dP3[i],2) - EXPT((length/2),2)) * ((GcodeGeoInfo.piStartPos[i]. dY - GcodeGeoInfo.piDestPos[i]. dY)/length);
        y_center := y_mid + SQRT(EXPT(Gcodegeoinfo.dP3[i],2) - EXPT((length/2),2)) * ((GcodeGeoInfo.piDestPos[i]. dX - GcodeGeoInfo.piStartPos[i]. dX)/length);
    END_IF
    angle := (ASIN(length/(2*GcodegeoInfo.dP3[i]))) * 2);
    GCodeGeoInfo.dLength[i] := angle*GcodegeoInfo.dP3[i];
    GCodeGeoInfo.dp1[i] := GCodeGeoInfo.dp1[i] := x_center;
    GCodeGeoInfo.dp2[i] := GCodeGeoInfo.dp2[i] := y_center;
IF(GCodeGeoInfo.iMoveTyp[i]=SM3_CNC.CLW) THEN
    GCodeGeoInfo.dT2[i] := 180 - 180/2)
    ELSEIF(GCodeGeoInfo.iMoveTyp[i]=SM3_CNC.CCLW) THEN
    GCodeGeoInfo.dT2[i] := -180/2)
    END_IF
ELSE
    //Calculation of dT1 when the center is at the origin
    IF(GcodeGeoInfo.dp1[i] =0 AND GcodeGeoInfo.dp2[i] =0) THEN
        IF(GcodeGeoInfo.piDestPos[i-1]. dX<0 AND GcodeGeoInfo.piDestPos[i]. dY>0) THEN
            GCodeGeoInfo.dT1[i] := 180;
        ELSEIF(GcodeGeoInfo.piDestPos[i-1]. dX>0 AND GcodeGeoInfo.piDestPos[i]. dY<0) THEN
            GCodeGeoInfo.dT1[i] := 0;
        END_IF
        IF(GcodeGeoInfo.piDestPos[i-1]. dY>0 AND GcodeGeoInfo.piDestPos[i]. dX>0) THEN
            GCodeGeoInfo.dT1[i] := 90;
        ELSEIF(GcodeGeoInfo.piDestPos[i-1]. dY<0 AND GcodeGeoInfo.piDestPos[i]. dX<0) THEN
            GCodeGeoInfo.dT1[i] := 270;
        END_IF
    ELSE
        //Calculation of dT1 for center other than origin
        IF(i>1 AND GcodegeoInfo.iMoveTyp[i] = SM3_CNC.CLW OR GcodegeoInfo.iMoveTyp[i] = SM3_CNC.CCLW) THEN
            a :=GcodeGeoInfo.piStartPos[i]. dX-GcodeGeoInfo.dp1[i];
            b :=GcodeGeoInfo.piDestPos[i-1]. dY-GcodeGeoInfo.dp2[i];
            IF (a<0 AND b<0) THEN
                GCodeGeoInfo.dT1[i] := SMC_ATAN2(Abs(b), Abs(a)) * 180/2)
            ELSEIF (b<0 AND a>0) THEN
                GCodeGeoInfo.dT1[i] := 360+(SMC_ATAN2(b,a)) * 180/2)
            ELSEIF (a>0 AND b>0) THEN
                GCodeGeoInfo.dT1[i] := SMC_ATAN2(Abs(a), Abs(b)) * 180/2)
            ELSEIF (a<0 AND b>0) THEN
                GCodeGeoInfo.dT1[i] := 360-SMC_ATAN2(b,a) * 180/2)
            ELSEIF (a>0 AND b<0) THEN
                GCodeGeoInfo.dT1[i] := 180-SMC_ATAN2(b,a) * 180/2)
            END_IF
        END_IF
    END_IF
ELSIF(GcodeGeoInfo.piDestPos[i-1].dY=0 AND a>0) THEN
    GCodeGeoInfo.dT1[i] := 360-(SMC_ATAN2(ABS(b), ABS(a)) *180/pi);
ELSE
    GCodeGeoInfo.dT1[i] := (SMC_ATAN2(b, a) *180/pi);
END_IF

ELSIF (i=1 AND Gcodegeoinfo.iMoveTyp[i] = SM3_CNC.CLW OR Gcodegeoinfo.iMoveTyp[i] = SM3_CNC.CCLW) THEN
    a := GcodeGeoInfo.piStartPos[i].dX - GcodeGeoinfo.dP1[i];
    b := GcodeGeoInfo.piDestPos[i].dY - GCodeGeoInfo.dP2[i];
    GCodeGeoInfo.dT1[i] := SMC_ATAN2(b, a) *180/pi;
END_IF

END_IF

AutomataGeoInfo[i].dLength := GCodeGeoInfo.dLength[i];
AutomataGeoInfo[i].dP1 := GCodeGeoInfo.dP1[i];
AutomataGeoInfo[i].dP2 := GCodeGeoInfo.dP2[i];
AutomataGeoInfo[i].dP3 := GCodeGeoInfo.dP3[i];
AutomataGeoInfo[i].dT1 := GCodeGeoInfo.dT1[i];
AutomataGeoInfo[i].dT2 := GCodeGeoInfo.dT2[i];

ELSE
    SMC_CalcLengthGeo(ADR(AutomataGeoInfo[i]));
END_IF

END_FOR

AutomataPath.nWritePos := Gcodegeoinfo.iObjNo;
AutomataPath.nCapacity := Gcodegeoinfo.iObjNo+3;
AutomataPath.bEndOflist := TRUE;
Automatapath.xPartofLM := TRUE;
END_IF

The above code helps in fetching the Gcode values from the CNC editor and converting into the format of the GeoInfo parameters. This necessity of the conversion arises because the interpolator can only interpret the GeoInfo parameters. Once these parameters are stored in a variable of the type GeoInfo, it is given as an input to the SMC_Interpolator function block and the device is logged in for the execution of the path.

These steps are to be followed only when the application has to run in the automatic mode. For the robot to get to a desired position when it is far from the initial position or if the robot has to start from the position that the user requires, there is a provision of jogging the drives that does not require the direct interpolation or the conversion of the position in the axis.
This function block is useful to help the user to give the direct input for the operation of the motors and drive them for the corresponding movement as per the requirement. The axis which is to be driven is given as a direct input to the input parameter “Axis” of the block and the JogForward and the JogBackward parameters are fed with the parameters/keys that are used as the activation parameter. These parameters are the one in the database with the unique address and are the buttons in the HMI which are used to move the drivers in the forward or the backward directions. The parameters of the velocity acceleration deceleration and the jerk are completely user dependent and can be defined as per the users’ choice and needs, depending on the amount of speed required while jogging of the motors, keeping on mind the parameters of the motors.
4.9 Import of DXF instead of writing the Gcodes manually.

Listing of the Gcodes can be difficult for the user to develop a particular motion paths for some precise and complex movements. This can be substituted with an alternative by designing the same path in the DXF format in any compatible software like AutoCAD and then trying to import the DXF files directly into the HMI workspace. This feature helps the user to make the path creation faster and can be flexible according to the requirements.

STEP 1:

This involves the import of the DXF by the user from the corresponding option in the HMI.

From the list of all DXF that is populated on the screen, we can select the desired file which is required for the path, and import to the HMI.
STEP 2:

The import of the specific file displays the preview of the file in the given workspace as shown:

The preview that is displayed on the screen is the exact path that the robot has to follow for the execution of the code. The overall path involves various geometrical shapes with respective values of movement which are crucial for the precise movement of the drives.

Once the import is completed, this path is automatically translated to the corresponding table in the HMI where the details of the motion for the movement along the X, Y and Z axis with the desired Feed rate and the radius for the curves and arcs are given. These data are transferred for the calculation of the internal parameters and conversion of these values into the GeoInfo format for the interpolator to interpret the details of the motion.
STEP 3:

This is the final visualization of the path along with the corresponding coordinates of the X Y and Z parameters that are read from the DXF file directly after import. The extraction of the parameters from the HMI is coded in the external routine of the dynamic link library file, that is called during running the application.

The data populated in the HMI are the values which indicate the movement of the axis in different axis with specific Feed rate which is the velocity by which the axis has to complete that specific path. For every path there is a start and the end velocity, along with the acceleration and deceleration values decided by the user. In our application, we have considered the acceleration and deceleration of the motion to be the half of the overall velocity. However, the acceleration and deceleration of the drive can be also indicated in the Gcodes directly, using the corresponding representation using the character ‘E’, where E+ indicates the acceleration and E- indicates deceleration in the path.

For the import of the DXF file from the disk, the corresponding command is used from the HMI which is linked to a dedicated external routine. This external routine has a case which is called when the import DXF option is selected by the user. The following line of codes help to populate the DXF file from the drive to the HMI for the users’ selection:
Once the required DXF file is imported by the user, there is an internal conversion of the data which takes place in order to convert the value format of the DXF to the .mod file of the HMI, which helps to populate the data in the user interface. This calculation is done in the visual studio using the C++ algorithms, which includes the procedure right from the opening of the file till the reading of the data based on the type of path processed in the DXF.

The sample algorithm for the accessing of the data can be shown as given below:

```cpp
void DxfDrawing::ExportEntities(std::vector<DxfEntity*>& entities, DxfInsert* insert)
{
    GasketingPrg.path.clear();
    Gdiplus::Color penColor;
    Gdiplus::Pen pen(Gdiplus::Color(255,255,255), 2.0);
    CGasketingABBMovement DxfMove;
    CGasketingABBProgram DxfPrg;
    Gdiplus::PointF lastPoint, curPoint, firstPoint;
    Gdiplus::REAL lastBulge = 0;
    for ( std::vector<DxfEntity*>::iterator shape = entities.begin(); shape != entities.end(); ++shape)
    {
        DxfLine* line = (DxfLine*)*shape;
        DxfMove.action = 0;
        DxfMove.move.zone = 0;
        DxfMove.wait = 0;
        DxfMove.move.um = GasketingFeedrate_mm_sec;
        DxfMove.move.speed = 200;
        DxfMove.ToPoint.w = 0;
        DxfMove.ToPoint.p = 0;
        DxfMove.ToPoint.r = 0;
        switch ((*shape)->Type() )
        {
            case TDxfLine:
            {
                DxfMove.type = MoveL;
                DxfMove.ToPoint.x = line->x1;
                DxfMove.ToPoint.y = line->y1;
                DxfMove.ToPoint.z = line->z1;
                GasketingPrg.path.push_back(DxfMove);
                if(line->x2!=line->x1)
                {DxfMove.ToPoint.x = line->x2;}
                if(line->y2!=line->y1)
                {DxfMove.ToPoint.y = line->y2;}
            }
```
if(line->z2!=line->z1)
{DxfMove.ToPoint.z = line->z2;}
GasketingFrg.path.push_back(DxfMove);
break;
}
case TDxfArc:
{
  DxfArc* arc = (DxfArc*)*shape;
  double StartX, StartY, EndX, EndY;
  Gdiplus::PointF center = TransformPoint((Gdiplus::REAL)arc->cx, (Gdiplus::REAL)arc->cy, insert);
  Gdiplus::RectF rect;
  StartX = arc->cx+arc->radius*cos(arc->angle1);
  StartY = arc->cy+arc->radius*sin(arc->angle1);
  EndX = arc->cx+arc->radius*cos(arc->angle2);
  EndY = arc->cy+arc->radius*sin(arc->angle2);
  rect.X = center.X - arc->radius * m_factor;
  rect.Y = center.Y - arc->radius * m_factor;
  rect.Width = rect.Height = arc->radius*2*m_factor;
  double start_angle = 360-arc->angle1;
  double end_angle = 360-arc->angle2;
  if(isArcClockwise(StartX,StartY,arc->cx,arc->cy,EndX,EndY))
  {
  DxfMove.type = MoveC_ccw;
  }
  else
  {
  DxfMove.type = MoveC_cw;
  }
  DxfMove.ToPoint.x = StartX;
  DxfMove.ToPoint.y = StartY;
  DxfMove.ToPoint.z = 0;
  DxfMove.move.zone = arc->radius;
  GasketingFrg.path.push_back(DxfMove);
  break;
}

These lines of codes are helpful for the extraction of the data from the file format of DXF and populating the corresponding values in the HMI of the user.

The different cases are useful for the different types of the drawing parameters used while creating the DXF file in the AutoCAD drawing. These various formats can be matched in the original drawing that is drawn in the AutoCAD and the corresponding case can be formed in the Visual Studio programming window.
4.10 **CALCULATIONS:**

The motors used for the movement in the specified path has its own configuration and it is required to understand the working characteristic based on the parameters that can be used for a normal functionality of the drive. Each drive used in the application inputs the data in the format of ‘count’, which is not easily understandable by the user in terms of the SI unit and hence it needs to be converted for the proper configuration to avoid the errors during the execution which may also lead to physical damage of the drives.

In most of the motors, the counts per turn is defined in the user manual and the physical data relating to the diameter and the gears are mentioned the configuration manual. These data are used for the corresponding conversion of the counts to the desired SI unit by the user.

Typically,

One motor turn = 360000 counts (total counts).

One motor turn = $2 \pi R$, where $R$ represents the radius of the drive. This gives the distance moved by the rotor in one revolution, or in 360000 counts.

Similarly, to calculate the maximum count and the maximum speed in meter per second, the following calculations are used:

$$\text{maxcount} = \frac{\text{Totalcounts}}{\pi \times \text{ShaftDiameter}} \times \text{stroke}$$

$$\text{maxspeedMPS} = \frac{\text{maxRPM} \times (\pi \times \text{ShaftDiameter})}{60000}$$

$$\text{motionvelocity} = \frac{\text{FwoBkvelocity} \times 60000}{\pi \times \text{ShaftDiameter}}$$

where,

- maxcount = maximum count corresponding to the stroke of the movement, which is limited as per users’ choice.
- maxspeedMPS = maximum speed that the motor should get in meters per second, based on the maximum RPM that the user inputs based on the motors’ configuration. The maximum speed of the motor can be obtained from the speed torque characteristics of the motor which is shown in the below
These characteristics can be obtained from the specific drives’ technical specifications obtained from the manufacturer.

While implementing the application in the real time Cartesian robot, the following calculations were used to calculate the various parameters of the motors:

Synchronous rpm \( (n) = f \cdot \frac{60}{p} \); where \( p \) is the pole pair number.

Synchronous speed \( (v) = 2 \cdot f \cdot \tau_p \); where \( \tau_p \) is the pole pitch.

Rotational frequency \( (w) = \frac{2 \pi n}{60} \); where \( n \) is the rotational speed in rpm.

FIG 4.4 Plot of the Torque Vs speed to determine the max sped of the motor
Alternate means of obtaining the characteristic of the speed of motor and the torque plot is by running the motor at different speed and observing the plot in the Sercos monitor, which also includes the cases of jittering and helps to obtain the right value for the position loop KV value which is the Proportional value in the PID controller.

In the above plot, we can observe a speed of motor up to 2000 rpm for the Nanotech AS5918S2904-E motor at a torque of 0.20 Nm.

As we run the motor at no load in the testing purposes, this speed can be achieved easily, provided there are not jittering or the resistance of noise and vibrations.

![Control diagram of the position inputting the calculated parameters in the loop for the set and Actual positions](image)

FIG 4.5 Control diagram of the position inputting the calculated parameters in the loop for the set and Actual positions

All the calculations obtained can be used also for the position control loop of the drives and can help in reducing the jittering of the drives. The feedback position control loop of the drive depends on the set position and the actual position of the drive where set position is the main input and the actual position indicates the current position of the drive. The difference between the two implies the error in the motion which needs to be compensated for a precise motion along the given path.
This describes the position control of the motion, where user can adjust the value of Kp according to the need of the motion. The value of Kp affects the jittering of the motor, and hence creates the vibrations. Therefore, to reduce this effect of the vibration, the Kp value should be chosen carefully and set according to the real-time application in the Robot.

The jittering and the vibration in the motors are caused due to the lower KV values defined. In order to have a smooth running of the motors, the configuration of the motor should include a KV value of around 65-75. This value was found out by running the motors in different configuration and observing the response of the sending and the receiving of the telegrams by the bus communication in the sercos monitor.
CHAPTER 5

REALTIME IMPLEMENTATION

5.1 SUMMARY

The execution of all these algorithms has to be implemented on a real-time system where the desired application had to run. The real-time implementation is done using the 3D Cartesian robot, with 3 axis. The three axis are given movement using the three respective motors of higher configuration in order to drive the axis in required path.

A Cartesian coordinate robot (also called linear robot) is an industrial robot whose three principal axis of control are linear (i.e. they move in a straight line rather than rotate) and are t right angles to each other. The three sliding joints correspond to moving the wrist up-down, in-out, back-forth. Among other advantages, this mechanical arrangement simplifies the Robot control arm solution. Cartesian Coordinate robots with the horizontal member supported at both ends are sometimes called Gantry robots; mechanically, they resemble gantry cranes, although the latter are not generally robots. Gantry robots are often quite large.

A popular application for this type of robot is a computer numerical control machine. And hence, we use these robots for such applications like Gasketing which are also a part of the Numeric control but using the optimum economic technique of SoftMotion where everything that a user does in a hardware of a NC machine is done directly through the PC integrated control drivers and programming within the CoDeSys.

For driving a Cartesian robot of a high dimension, it requires the motors which can drive them with the right amount of torque and maintain the feed supplied at the input of the drive. For the tests that were performed to check the real-time behavior of the system on using the SoftPLC in the drive, for the motion, we have used the following configuration of the motors and the drives.
Motor type – MSK servo motor

Configuration details:

- Type - MSK060C-0600—NN-S1-UG0-NNNN
- Max Torque – Up to 495 Nm.
- Max Speed – Up to 9000 rpm.
- Encoder system for a wide and diverse range of applications.
- High protection category IP65
- Choice of the cooling systems.

Step by step guide is as shown below:

**MSK060C-0600-NN-S1-UG0-NNNN**

- **Motor:**
  - Size
  - Overall Length
  - Winding
- **Cooling System:**
  - NN = Natural Convection (Surface cooling or option for certain sizes)
- **S1 = Single turn Encoder 128 increments**
- **Shaft:**
  - G= Plain shaft with shaft sealing ring
  - P= Keyway confirming to DIN 6885-1 and shaft ring.
- **Other Versions:**
  - N = Standard
  - S= Hazardous duty version
- **Holding Brake:**
  - 0: Without holding brake
  - 1: With electr. Released holding brake
  - 2: With heavy duty holding brake
  - 3: With extra heavy duty holding brake
The detailed technical data of the motor used for the implementation of the application can be found in the appendix [1]. The Robot that was used in the application test was the following:

![3D Cartesian Robot](image)

**Fig 5.1 The 3D Cartesian Robot where the application was implemented**

The two axis of the machine is driven by a dedicated motor of the above configuration which is in turn having a dedicated driver driven by a PAC coded through the CoDeSys V.3.5.

The movement of the X Y and the Z axis is observed in the output through end effector which has the overall movement of the robot. This movement can be observed on a plane by just depicting it as a drawing of the movement of the end effector in the prescribed path with the required precision.

The mechanical configuration of the robot especially for the dedicated axis X Y and Z is represented in the appendix [2], [3] and [4] respectively.
The movement of the end effector can be observed through the below image:

![Image of end effector and Z Axis]

**Fig 5.2 Z Axis and the end effector path movement**

From the movement of the end effector, we can compare the input and the reciprocation of the path by the robot. Once the path is inputted to the drive, the motion is begun in the three axis of the robot, and the overall path is generated by the motion of the robot which can be observed on the motion area after the movement is completed.
ANALYSIS AND CONCLUSION

SUMMARY

The revolution in the field of industrial Automation is constantly increasing and the demand for the development of the cost effective techniques for the implementation for the various mechanical applications are growing high exponentially at every level. This was the reason for the cost reduction technique using the controllers and the PC based NC control through a dedicated HMI and the SoftMotion technique of CoDeSys, in the manufacturing process. This procedure was adopted in the motion control and the path following technique, where the application tester was a 3D Cartesian Robot. The results observed were very precise, based on the motion that was given to be followed and the path that was actually followed by the end effector.

The development of this control application linking the user interface with the Modbus using a dedicated PC of the user was a very cost effective in terms of the implementation and execution. The dedicated PACs and the drivers for the motors are the only investment that are involved to observe such implementations in the mechanical industry, whereas the NC machines involve a high investment and maintenance problems.

FUTURE DEVELOPMENTS

Considering the high level application of such machines and drives, in future, there are the possibilities of carrying forward the same technology to implement the same applications to the Industrial Revolution 4.0, where the application would just be needed to installed and run on the PCs based at the far off places where similar application has to run.

This would make the control and operations simpler and would help to develop the technology in a more reliable way in terms of implementation.
APPENDIX

[1]: The description of the motor used in the 3D Cartesian Robot for the real time implementation of the application.

<table>
<thead>
<tr>
<th>Motor</th>
<th>Maximum speed</th>
<th>Cont. torque at standstill</th>
<th>Maximum torque</th>
<th>Cont. current at standstill</th>
<th>Maximum current</th>
<th>Moment of inertia</th>
<th>Dimensions</th>
<th>Mass</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$n_{\text{Max}}$ [rpm]</td>
<td>$M_{0\text{.6}K}$ [Nm]</td>
<td>$M_{\text{Max}}$ [Nm]</td>
<td>$I_{0\text{.6}K}$ [A]</td>
<td>$I_{\text{Max}}$ [A]</td>
<td>$J_R$ [kgm$^2$]</td>
<td>A [mm]</td>
<td>B [mm]</td>
</tr>
<tr>
<td>MSK060C-0300-NN-S1-UG0-NNNN</td>
<td>4900</td>
<td>8</td>
<td>24</td>
<td>4.8</td>
<td>19.2</td>
<td>0.0008</td>
<td>116</td>
<td>226</td>
</tr>
</tbody>
</table>


[3]: Axis X-Y of the 3D Cartesian Robot for Gasketing
[4]: Axis Z of the 3D Cartesian Robot for Gasketing
REFERENCES

1. Flexible motion control using IEC 61131-3.
   M. Rayment

2. Design and verification of mechatronic object-oriented models for industrial control systems.
   M. Bonfe; C. Fantuzzi

   Valeriy Vyatkin; Martin Hirsch; Hans-Michael Hanisch

   K.K. Tan; S.Y. Lim; S.N. Huang

5. https://wiki.sercos-service.org/current/Main_Page


7. Commanded and coordinated motor axes in an IEC 61131-3 application
   Patrick Rohloff, 3S Smart Software Solutions

8. A CNC system based on STEP-NC and Function Block architecture
   Mohamad Minhat and Xun Xu; Valeriy Vyatkin

9. An Open Architecture Numerical Control System Based on Windows CE
   Suhua Xiao; Di Li

10. Theory and design of CNC systems.
    Suk-Hwan Suh, Seong Kyoong Kang, Dae-Hyuk Chung, Ian Stroud