CODEN:LUTEDX/(TEIE-5423)/1-68(2019)

Industrial Electrical Engineering and Automation

Virtual Commissioning for a Linear 12-Axis Machine

Eric Chronvall
Christian Svensson

Division of Industrial Electrical Engineering and Automation
Faculty of Engineering, Lund University

Virtual Commissioning for a Linear 12-axis machine

Eric Chronvall
Christian Svensson

June 2019

LUND
UNIVERSITY

Beckhoff : *Daniel Jovanovski*
Fredrik Malmgren

AP&T : *Christer Bäckdal*

Division of Industrial Electrical Engineering and Automation
Lund University
Box 118
SE-221 00 LUND
Sweden

© 2019 by Eric Chronvall and Christian Svensson.
All rights reserved.
Lund 2019

Abstract

In practice, Virtual Commissioning is a technical method within the automation field where you can create a simulated twin factory to a real physical factory containing functional machines. The purpose of Virtual Commissioning is to move a great portion of the commissioning tasks to an earlier state in the project. To make this possible a simulation model of a system is created within a simulation program. The behavior of the system is then programmed through an external PLC program. This allows for software testing simultaneously with the software development environment to verify the functions of the PLC program.

This thesis investigated the possibility of using a simulation program, Visual Components, on four transportation robots with three linear axes each. These transportation robots would then together represent the entire linear 12-axis machine. The four robots could then be moved separately or all together by changing their 3D-positions through PLC code sent into Visual Components. Furthermore the programming environment of TwinCAT was used. The vision was to connect Visual Components and TwinCAT through a communication protocol - Beckhoff ADS. Through ADS communication, data could be sent back and forth between the two programs. To structure the code, the OMAC PackML construction was decided to be implemented. In addition, a HMI (a display with buttons to be pressed by a machine operator) was constructed. By the HMI, all necessary functions such as point-to-point movement, jogging and stopping were implemented. The final result was a software program based upon PackML in TwinCAT, a HMI and a Visualized model in Visual Components.

Acknowledgements

We would like to give sincere appreciation to Beckhoff Automation AB and their staff, especially within the technical support department. A special thanks to Daniel Jovanovski for giving us fantastic guidance throughout the project and Fredrik Malmgren for support with documenation and project planning. A thanks to Fredrik Nygren and Mattias Nilsson who supported us with knowledge within OMAC PackML.

We are sincerely grateful for the support of AP&T making this thesis a possibility. Especially Christer Bäckdal, vice CTO of AP&T, not only assigning us this task but giving us valuable advise in order to accomplish this project.

We are grateful for the support of our supervisor Gunnar Lindstedt at IEA, dedicating valuable time to us in time of need. Furthermore, we would like to give appreciation to Ulf Jeppsson, taking the role as examiner for our project and giving valuable feedback.

Every input from these people have been of great importance for us to complete this thesis. Additionally, we would like to thank Beckhoff who lend their PLC equipment to us which provided us a realistic hardware setup.

We feel grateful for the CAD-file of the transportation machine AP&T gave us in order to create a simulation. At last, a thanks to Visual Components for providing us with two free "Trial Licences". Without the equipment, the CAD-file and the licenses this project would not have been possible to pursue.

Foreword

The authors of this project have a background in the field of electrical engineering (specialised in automation and control theory), as well as interest in its industrial application. Therefore the automation topic Virtual Commissioning could potentially be a rewarding and developing subject to learn more about. A possible description of Virtual Commissioning is a 3D-simulation model for one or several interacting 3D-objects. These objects are supposed to visualize a systems behaviour. By this technology, adjustments and upgrades in code can be both tested out safe and be observed before implemented into a real system [Searcherp].

The task at hand is to evaluate how Virtual Commissioning can be integrated into Beckhoffs automation software and tested on a 12-axis machine, which is the scope for this Master thesis [Assemblymag].

The work-load of the project was divided fairly between the authors of this project. Eric focused on implementing a HMI (Human Machine Interface) - which was a visual display of all machine functionalities with push buttons. These buttons could be start, stop, move or reset the machine which were pushed from the HMI. Christian focused on coding the PackML structure and alarm handling in TwinCAT. The PackML structure was how the code should be organized and the alarm handling was the code for dealing with potential software-errors. Together, the authors created a EtherCAT safety project in TwinCAT which was connected to a physical "emergency stop" button and a "safety gate" in the 3D-simulation program. Furthermore, the authors helped out creating a simulation file from a CAD-file of the 12-axis machine in Visual Components.

Nomenclature

ADS = ADS is based upon the communication principle "client to server". Basically, on a network the "server" has functionalities and data which "clients" have to ask permission for to access. Therefore, the "client" have to send a request to the "server". Afterwards, the "server" reply if it can share the server data. In that case, a communication path-way is built between the "client" and the "server".

AmsNetId = An ADS IP-adress with 48 bits which makes it simpler to find ADS supported devices. It is basically a IPV4 adress where ".1.1" has been added to the end of the address.

Contactor = An electromagnetic switch which blocks or opens the current flow inside an electrical circuit.

CPU = A Central Processor Unit (CPU) is the main device in a computer which executes a program based upon data it understands.

CRL = A list within the ADS-protocol of a device which keeps track if an ADS address is correctly configured into another device with support of ADS. Otherwise, this list will generate a corresponding error message to the fault that occurred.

Dynamic telegram exchange = Ethernet frames could reach a multiple of devices in both send and receive direction of a telegram.

ETG = EtherCAT Technology Group is a firm which provides devices for communication system purposes. ETG currently has the most employees in this branch in the entire world.

Fieldbus Memory Management Unit = The memory manager of a fieldbus which controls how data is allocated in the memory space.

FSoE = A communication protocol over EtherCAT which can send and receive safety and control-packages.

GVL = Global variable within a global variable list in the PLC-environment.

I/O:s = The inputs (I) and outputs (O) from the physical PLC device.

Integrated Development Environment = An integrated development environment is a software program that provides a platform for computer programmers to develop programs in.

MAC = An Ethernet MAC (Media Access Controller) uses a data address which purpose is to specify the destination and the source of each data-packet sent on a network.

NC = A controller implemented in a computer for processing numerical data in different applications.

TCP/IP = A communication protocol that is based upon sending data packages through an internet connection, by Ethernet or a wireless such.

UDP/IP = A communication protocol that is based upon sending real-time data packages without internet connection.

Virtual Commissioning = A 3D-simulation model of a system is created which represents the real factory behaviour..

Visual Components = A 3D simulation program which establish a connection to PLC software through a ADS-communication.

Contents

1 Introduction

The last step in the engineering process is usually commissioning. The commissioning of an automation system project can take around 15-20% of the total delivery time. It has been shown that almost 2/3 of the time spent in commissioning is spent on fixing software errors. Firstly, this is because all the hardware needs to be produced, which then can be followed by testing of the control software. Often the commissioning is performed under very strict deadlines which can lead to rushing the development of the control software. This can result in bugs in the system, that can yield hardware damage.

The idea of Virtual Commissioning is to move a great portion of the commissioning tasks to an earlier stage in the project. To make this possible a simulation model of the system is created which represents the real factory. The real control system can then be used on the virtual factory. This allows for software testing simultaneously with the development to verify the functions of the program. It also makes error detection quicker. Furthermore it could be used as a "Digital Twin" as well as in education purposes [Assemblymag].

In this project a 12-axis machine for transportation of objects, provided by APT, is simulated through a program called Visual Components. The machine consist of four transportation robots. The functionality of the machine will be coded in Beckhoff's TwinCAT PLC program. The structure of the code will be implemented according to the PackML-standard.

AP&T

AP&T is a company that develop high-end production solutions for the metal-forming industry since the 1960:s. It is a global company that accounts for sustainable solutions towards their customers. These customers are mainly in automotive, energy and roof industries. Its main office is located in the town of Ulricehamn in Sweden. The companys arsenal of products consists of production machines, presses, metal tools and services for retailers of metal-forming. APT:s goal is to achieve the highest possible satisfaction and safety for its customers [AP&T_Company].

Beckhoff

Beckhoff is an automation company that has its origin in Germany. Today it is a worldwide company with the headquarter located in Verl, Germany. Beckhoff implements open automation system based on PC control technology. It manufactures products such as industrial PCs, , I/O:s, drive technology and automation software. Beckhoff's market idea is to support global and open control and automation solutions. The application of their products could be all from CNC-machines to automation for buildings [Beckhoff_Company].

2 Theory

2.1 The 12-axis machine of AP&T

The 12-axis machine is a transfer-machine of metal objects. In comparison to other similar machines on the market, the energy consumption of this machine is told to be 50 % less. One functionality is that the control unit to this machine can be maneuvered by smartphones or tablets. The machine also has support for observation of its status and statistics regarding production. Furthermore, this machine can be programmed to move around in a specified amount of axis directions in 3D-space. The movement itself is a linear one, from one point to another, so called PTP (Point-To-Point) [AP&T_Automation].

2.2 Visual Components

Visual Components is a 3D-simulation program which belongs to the top tier within their branch. The main goal of this company is to increase availability and simplify the use of its own products towards its customers. The program can take CAD-files (computer drawings) and convert them into simulation files which can be programmed to move in X,Y and Z direction in 3D. This can for instance be the 12-axis machine mentioned previously (see Section 2.1) (see Figure 1).

In order to communicate with another program, Visual Components uses ADS (Automation Device Specification) (see Section 2.5) for this to work [Beckhoff_ADS] (6). As previously mentioned, a "client-server" connectivity has to be established. One typical application could be ADS connectivity between a PLC-software program and [VisualComponents_ConnectPLC].

Currently, Visual Components holds a position as a primary trademark within simulation software for industrial applications among automation brands. Two brands who can integrate this software with their own are Beckhoff and Siemens.

The office headquarter is located in Finland, in the city of Espoo. Secondary offices are located in USA and Germany. Otherwise, they have business-partners and retailers located all around the world [VisualComponents_About].

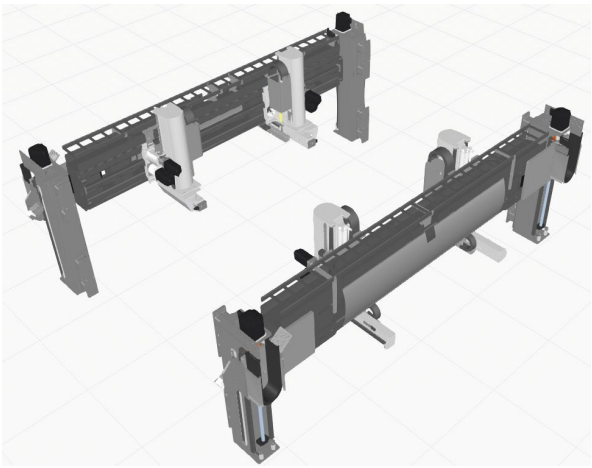

Figure 1. Linear 12-axis machine simulated in Visual Components.

2.3 Virtual Commissioning

The meaning of Virtual Commissioning is somewhat self-explanatory. Virtual means simulated on a computer or a computer network. Commissioning includes designing, installing, testing, operating and maintaining industrial systems according to the requirements. As mentioned in the introduction the commissioning can take 15-20% of the total delivery time. Any method of reducing this time would therefore be of interest for the industry.

A large portion of the commissioning (60-70 %) is actually spent on fixing software errors. By moving the commissioning tasks to an earlier state it increases the chances of avoiding mistakes by simply running tests as the code is written [Siemens]. This makes it possible to reduce the overall time for the commissioning. Another aspect is that this method also reduces the risk of hardware damage. In traditional commissioning the software has to be tested on the real machine. If the software would contain any bugs it could result in damage on the machine. By using Virtual Commissioning this can be avoided by observing the behaviour of the simulation and prevent any hardware damage from happening [VisualComponents_VC] . Furthermore it can be used to create a "Digital Twin" which would allow for components to be replaced by the digital replica. This could, for example, be replacing sensors upon breaking. It can also be used in education purposes.

2.3.1 System Structure for Virtual Commissioning

The system structure for Virtual Commissioning consists of four parts. Control system, Plant, Simulated Plant and Simulated control system. This structure and its communication between the different parts enable a fully functional realization of an industrial factory (see Figure 2) [CTH] (11).

The Control system's purpose is to provide control data sent from the physical PLC device to the Plant and the Simulated plant [Beckhoff_WindowsControl]. For this to work, the same control data has to work with both the plant and its simulation [VisualComponents_VC] (10).

The Plant represents the real industrial factory with all the machines and devices that are connected to it. The commissioning of a plant could imply manufacturing of products within production lines. Production lines mean the chain of machines that in a chronological manner carry out jobs to finish a product [CollinsDictionary].

The Simulated control system could for instance represent an EtherCAT (see Section 2.8) simulation which runs I/O:s through an EtherCAT cord. When this happens I/O-information can be sent to represent real machine data. That data could represent signals like motor or sensors connected to machines [Beckhoff_ECATSimulation].

The Simulated plant is supposed to imitate the real factory plant behaviour by a 3D-visualization. Here software tests can be done by simply coding or adjusting I/O values so the behaviour resembles a real plant [VisualComponents_VC].

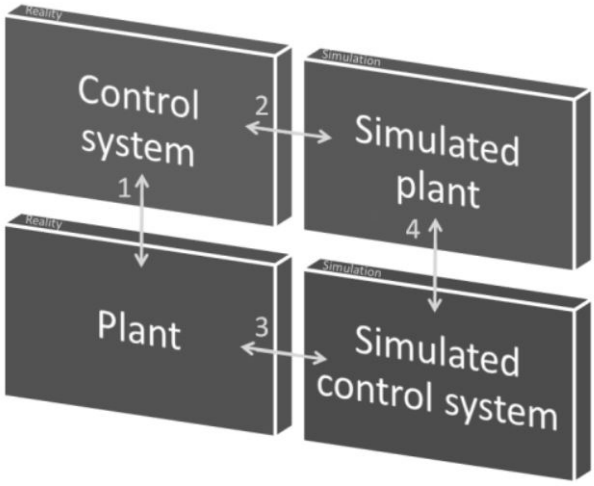

Figure 2. The system structure of Virtual Commissioning.

2.4 ADS Communication

The ADS protocol uses the transport layer inside a software program of a device to communicate with other devices holding ADS. The purpose for this protocol is data management between different software programs on one or several devices. The protocol enables whatever device on a connected network to interact at any point in time.

The principle behind ADS is called client-server. Firstly, an ADS request is sent out from the client device on the network. Eventuelly a server device reacts by an indication message. Secondly, the server sends a response message to the client device and after that a communication pathway is built. Once the pathway has been constructed, data traffic commences between the client and the server [Beckhoff_ADS].

2.5 Visual Studio

Microsoft Visual Studio (VS) is an IDE (Integrated Development Environment) provided by Microsoft. Its area of use is to develop websites, web apps, web services, mobile apps as well as computer programs. To make this possible VS uses software development platforms from Microsoft such as Windows Forms, Microsoft, Silverlight, Windows API etc.

VS includes a code editor together with code refactoring. Furthermore the integrated debugger can handle both machine-level debugging and source-level debugging. VS is applicable for 36 different programming languages while allowing the debugger and code editor to support practically any programming language (provided that there exists a language-specific service) [VisualStudio_IDE].

2.5.1 Visual Studio Isolated Shell

In this project the Visual Studio Isolated Shell (2013) is used. It enables VS partners to build applications and tools on top of the VS IDE. While in integrated

mode, its possible to release a VS extension for customers to use (who have not yet installed VS). In isolated mode you can release a custom application that makes use of subset of the features of VS IDE [VisualStudio _IsolatedShell].

2.6 PLC

Simply put, a PLC is a programmable logical controller - a PC aimed for applications in industrial systems. Whenever there exist a need for control of devices the PLC can connect these devices' software together so they can perform tasks. A basic PLC uses a CPU (Central Processor Unit) that controls a number of inputs which uses logic to achieve outputs for the sought control. PLCs are programmed to be adaptable, have high robustness (rarely crashes or give mechanical havoc) and be cheap compared to similar devices [PLCdev].

The TwinCAT PLC is Beckhoff's main software platform for control oriented tasks. The software platform can make almost any PC into a real-time controller. TwinCAT includes a multi-PLC system, programming environment, operating station and a NC axis control. Up to four parallel PLC's can be run independently on the same time-window for one runtime system. The runtime system means the program which compiles the code from top to bottom with all given inputs and outputs. The time-window will then be the execution time for this compilation. Furthermore, the TwinCAT PLC follows the programming standard IEC 61131-3 [Beckhoff_E-CATG]. This standard enables up to five different programming languages to be used for the PLC. One common language is ST (structured text), which means that the code has to be written in a high level manner. In TwinCAT the high level functions consider mostly *if*, *case* and *for* -statements that could be used to write logical code [Beckhoff_PLC].

A typical PLC device used by Beckhoff is the CX5140. Its main characteristics is that it has low power consumption and is built without fans for cooling the device. This PLC can install software for TwinCAT runtime (XAR) and be usefull to both PLC or Virtual motion (simulate movement of virtual axes) applications. It has two gigabit supported Ethernet connections and also four USB 2.0 connections. The operating temperature of the PLC device ranges from -25 to +60 degrees in the surrounding environment [Beckhoff_CX5140] (see Figure 3).

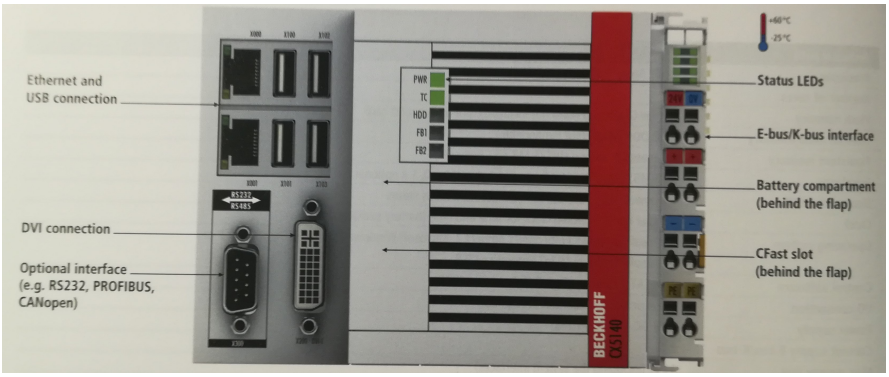

Figure 3. The CX5140 PLC from Beckhoff.

2.7 TwinCAT

TwinCAT is Beckhoff's own software which is used to develop programs. The TwinCAT software uses parallel real-time systems to manage control programs and the software development environment. TwinCAT executes programs based on an accurate time-and priority management. This provides a highly deterministic behaviour without interfering with the processor tasks.

Both hardware and software that are from open system PC's need valid control and support to prevent errors. Therefore TwinCAT has a built-in measurement tool, called Jitter-indicator, which keeps track of real-time Jitter that affects signals in the software [Beckhoff_WindowsControl]. When a signal has jitter it means that the signal has some sort of delay. The delay could for example be variations in phase or time-period of the signal [ElectronicDesign]. Once a jitter appears, the Jitter-indicator will send a system message in the software-program that informs the programmer about how and when it happen [Beckhoff_WindowsControl].

2.7.1 Object Oriented functions

The following sections will describe some functionalities that can be used in the TwinCAT software.

Object Function block

A function block is a data file that returns a number of values decided by the programmer when compiled. The function block is called upon by an instance, a new copy of the same function block. Function blocks can also implement object method and object action data types [Beckhoff_FunctionBlocks].

Object Method

The object oriented method consists of expressions in PLC code that has to be connected to either a function block or a program. A method takes input data through a call, computes a result and sends it back to the place it was called upon as output data. Data which is sent into a method is declared VAR_IN within the method. Data the method sends back is declared VAR_OUT within the method. The data which is compiled within the method exists only while the method runs [Beckhoff_Method].

Object Action

The object oriented action is a number of code lines (structured text) that only runs when called upon. The action does not have declarations and uses data created from function blocks or programs [Beckhoff_Action] .

Global Variable

A global variable is a variable or a constant that is defined in the entire solution of a project and can be used from all locations in the solutions code. These variables must be created in a global variable list (GVL) of the type

VAR_GLOBAL to be used. Writing to a variable can be done through call upon the name of a list followed by a "dot" and the variable name, for instance "GVL_Default.bSensor" [Beckhoff_GVL].

Struct

The data type struct is a construction of an object with one or more input variables [Beckhoff_Struct]. A way to see a struct is like an interface - an empty function-shell with inputs that once called upon can be implemented and given functionality [MIT]. An example of a struct could be a Polygon line with a few inputs; a starting point, intermediate points and an endpoint. To use a struct, it either has to be added from a PLC library or it has to be manually created within a PLC project [Beckhoff_Struct].

NC Axes

TwinCAT has numerous types within its Numerical control systems (NC), but for this project the focus is put on "slave axes" and "master axes". The "slave axes" functionality depends on values generated from the master axes they are coupled too. In spite of coupling or decoupling, these axes have no functions themselves. Instead they operate in linear response to the "master axis". The master axis deals with all movement; for instance start, stop and jog which the coupled slave axes will imitate [Beckhoff_NCAxes].

2.7.2 TwinCAT Motion Control Library

This section will describe the function blocks that are used throughout the project.

MC_Power

MC_Power makes motion control software enabled for an axis when the signal "Enable" equals true. Once enabled, by setting either or both "Enable_Positive" or "Enable_Negative" to true, it will make a virtual axis support movement in one or two directions. To make an axis controllable by other MC functions blocks, the "Override" signal must be set to a value above 0. Furthermore, the "Override" signal can be set between 0 to 100 % - how much the current velocity should be overridden (see Figure 4).

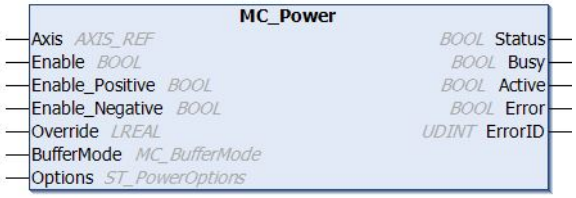

Figure 4. MC_Power function block.

MC_Reset

MC_Reset basically resets the virtual axis it is called upon. A common event is that connected devices to the axis also get reset. To enable the reset the input signal "Execute" needs to be set true. For turning the reset off, the "Execute" simply has to be set false. When the resetting is finished, an output signal "Done" will flag this by giving true (see Figure 5).

Figure 5. MC_Reset function block.

MC_MoveAbsolute

MC_MoveAbsolute sets a destination "Position" and moves towards it when an input signal "Execute" is true. Once the targeted position has been reached, the output signal "Done" gives a flag by outputting true. Parameters that have to be set to a defined value larger than zero if an NC axis should move is Velocity, Acceleration, Deceleration and Jerk. This function block can be used by several types of axis systems, mainly linear though. A useful functionality is that MC_MoveAbsolute can be used to move master to slave coupled axes (see Figure 6).

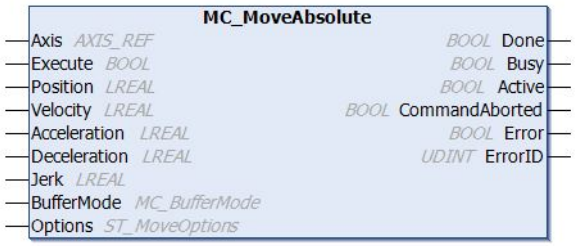

Figure 6. MC_MoveAbsolute function block.

MC_Halt

MC_Halt halts an axis movement by a braking procedure when "Execute" is set to true. However, this action will not lock the axis from moving. Once the output signal "Done" flags true - "Execute" can be set to false again. Parameters that has to be set to a defined value larger than zero if an NC axis should be halted is "Deceleration" and "Jerk". Fortunately, MC_Halt can be applied for master to slave coupled axes. Although, using this function block will uncouple the master to slave configurations. Therefore, the previously coupled axes have to be re-coupled again once MC_Halt has been called upon (see Figure 7).

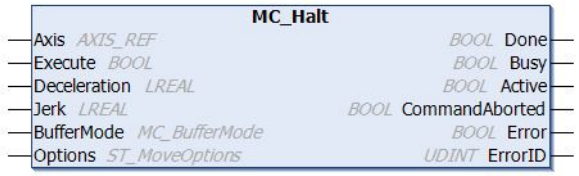

Figure 7. MC_Halt function block.

MC_Stop

MC_Stop forces a stop to an axis movement by turning all function blocks execute flags to false when "Execute" is set to true. This is different from MC_Halt which would not turn off execute flags of other function blocks. Unlike MC_Halt the MC_Stop will lock an axis from moving. Once the output signal "Done" flags are true - "Execute" can be set to false again. Parameters that has to be set to a defined value larger than zero if an NC axis should be stopped is "Deceleration" and "Jerk". MC_Stop can also be applied for master to slave coupled axes (see Figure 8).

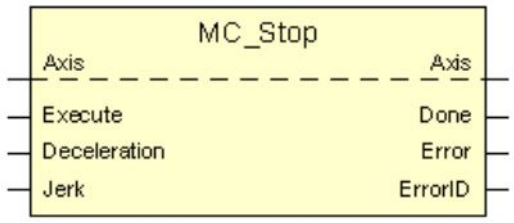

Figure 8. MC_Stop function block.

MC_GearIn

MC_GearIn uses a linear coupling between the master and slave axis. How big effect the master has on the slave is affected by the "RatioNumerator/ RatioDenominator" ratio. The higher the ratio, the more the slave will move relative its corresponding master and vice versa. To be noted, a slave can only be coupled to a halted/stopped master axis. Additionally, more than one slave can not be coupled to a master axis on the same time. The "InGear" output signal gives a true signal when a coupling is completed. Once a master to slave coupling has given "InGear" equal to true for the function block, the next slave can couple to the master (see Figure 9).

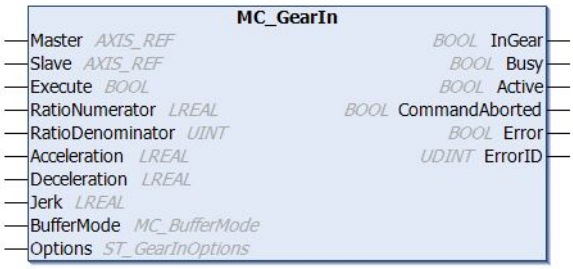

Figure 9. MC_GearIn function block.

MC_GearOut

MC_GearOut removes the current coupling between a master and its selected slave axes. When the "Execute" signal is set to true, the out-gearing commences and when the "Done" flag gives true, the out-gearing was successful. To make the function block ready for a new gearing, the "Execute" signal should be set false again (see Figure 10) [Beckhoff_Motion].

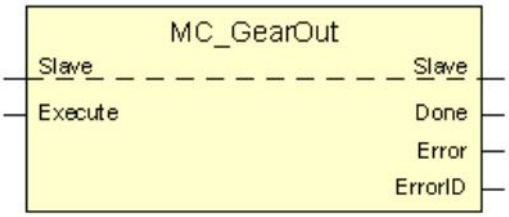

Figure 10. MC_GearOut function block.

2.8 EtherCAT

The ETG (EtherCAT Technology Group) is a firm which provides devices for communication system purposes. ETG currently has the most employees in this branch in the entire world.

ETG are partners with IEC (International Electrotechnical Commission). EtherCAT and Safety over EtherCAT follows, IEC 61158 and IEC 61784, respectively [EtherCAT_Company].

For a long time Ethernet communication has had technical problems with efficiency in managing the exchange of data-packages in telegrams. These problems are resolved through EtherCAT communication. Nowadays, the Ethernet packages are no longer accepted, analyzed and copied as a work process at every time cycle. Instead, a Fieldbus Memory Management Unit (FMMU) for each I/O terminal analyses the data addressed to it, while a telegram continues through the device. In the same manner, input data enters the telegram while it passes through. This process is called Dynamic telegram exchange (described more in the following section). The delay-time for this exchange is a few nanoseconds in duration.

2.8.1 Functional Principle

First the EtherCAT master sends a telegram that passes through each node. Each EtherCAT slave device then reads the data that is addressed to it "on the fly", which means that they will read the data as the telegram is moving. When the data is read it gets inserted into the frame as the frame moves downstream. The only delay the frame experiences is the one from the hardware propagation. The last node in a branch/segment will detect an open port and send the message back to the EtherCAT master using Ethernet Technology's full duplex feature.

In terms of performance, the EtherCAT technology makes advanced control tasks a possibility which traditional field bus systems could not manage. An example is that Ethernet systems can handle both velocity and current control of drive systems simultaneously, instead of only velocity control. Since the bandwidth has increased by 80%, status updates between data-packages in telegrams can be achieved. Furthermore, 1000 I/O:s can communicate within 30 microseconds. Another feature is that the bus system is no more a bottle-neck of the system speed . Now the technology exists to actually reach speeds up to 10 000 Mbit/s [Beckhoff_E-CATFast].

The only node (within a segment) allowed to actively send an EtherCAT frame is the master. All other nodes merely forward the frames downstream. The purpose of this concept is that it prevents unpredictable delays and guarantees real-time capabilities.

The master is using a standard Ethernet MAC (Media Access Controller) without an additional communication processor. This makes it possible for the master to be implemented on any hardware platform which has an Ethernet port (regardless of which real-time operating system or application software is used). The slave uses an EtherCAT Slave Controller (ESC). This controller processes frames "on the fly" and entirely in the hardware. This allows network performance to be predictable and independent of the individual slave device implementation [EtherCAT_Function].

2.8.2 The EtherCAT Protocol

The payload of the EtherCAT is embedded in a standard Ethernet frame. The frame is identified in the EtherType field with the Identifier (0x88A4). The use

of protocol stacks, such as TCP/IP and UDP/IP, can be eliminated since the EtherCAT protocol is optimized for short cyclic process data (see Figure 11).

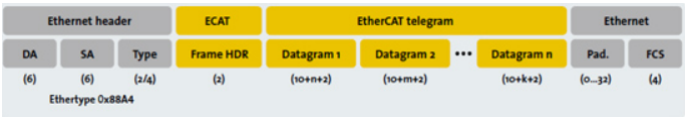

Figure 11. EtherCAT in a standard Ethernet frame.

TCP/IP connections can optionally be tunneled through a mailbox channel without impacting the real-time data transfer. This is done to ensure the Ethernet IT communication between the nodes.

During startup, the master maps and configures the process data on the slaves. It is possible for different amounts of data to be exchanged with each slave (from one bit to a few bytes, or even kilobytes of data).

The EtherCAT frame consists of one or more datagrams. The header of the datagram indicates what type of access the master would like to execute [EtherCAT_Function]. This could be:

- Read, write, read-write ;
- Access to a specific slave through direct addressing, or access to multiple slaves with logical addressing (implicit addressing);

2.9 Safety over EtherCAT

Modern communication systems are able to both realize the deterministic transfer of control data as well as enable the transfer of safety-critical control data through the same medium. With this very purpose EtherCAT utilizes the protocol FSoE (Fail Safe over EtherCAT). This protocol allows:

- A Single communication system both for safety and control data;
- The ability to flexibly expand and modify the safety system architecture;
- Pre-certified solutions to simplify safety applications;
- Logical integration of the safety design in the machine design;
- Powerful diagnostic capabilities for safety functions;
- The ability to use the same development tools for both safety and standard applications;

With Safety over EtherCAT, the communication is part of a "Black Channel". A "Black Channel" means that the communication medium, an EtherCAT cord for instance, sends data-packages that are both safe and not-safe. The standard communication system EtherCAT uses a single channel for the transfer of both standard and safety-critical data. The Safety Frames (known as Safety Containers), consist of safety-critical process data with some additional information used to secure this data. These Safety Containers are transported with the communication's process data. Whether the transfer of the data is safe or

not does not depend on the underlying communication technology and is not restricted to EtherCAT. The Safety Containers can be transported through fieldbus systems, Ethernet (or similar technologies), as well as make use of copper cables, fiber optics and even wireless connections (see Figure 12) .

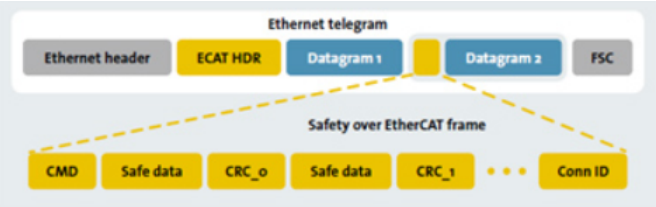

Figure 12. The Safety Container is embedded in the cyclical communication's process data.

The safety-connection of different parts of the machine becomes more simple with this flexibility. The Safety Container goes through the various controllers and is processed in the various parts of the machine. This makes the emergency stop functions easy to implement. Its simple to either bring the whole machine or targeted parts of the machine to a standstill. This is allowed even if parts of the machine are coupled with other communication systems (such as Ethernet) [EtherCAT_Function].

2.10 TwinCAT Safety (TwinSAFE)

TwinSAFE from Beckhoff enables convenient expansion of the Beckhoff I/O system with safety components in addition to integration of all of the cabling for the safety circuit within the existing fieldbus. It is possible to mix safe signals with standard signals without restrictions. The TwinSAFE telegrams are handled by the standard controller.

The TwinCAT Safety PLC is used to link safety-related inputs and FSoE outputs. TwinCAT Safety PLC realizes a safety-related runtime environment on a standard industrial PC.

ESTOP

The "emergency stop" (ESTOP) button can be connected to two normally closed contacts on a safe input terminal on an EL1904 safety-card (see Figure 13). The EL1904 safety-card contains all safety related software that needs to be implemented for EtherCAT safety. These types of safety cards all begin with "EL" and have similar functionalities and end with 4 digits; in this case "1904" which displays what model it is. The monitoring and testing of the discrepancy of the two signals are activated. The feedback and restart signals are connected to standard terminals and transferred to TwinSAFE through the standard PLC. The Contactor K1 and K2 are connected in parallel to the safe output [Beckhoff_SafetyCard].

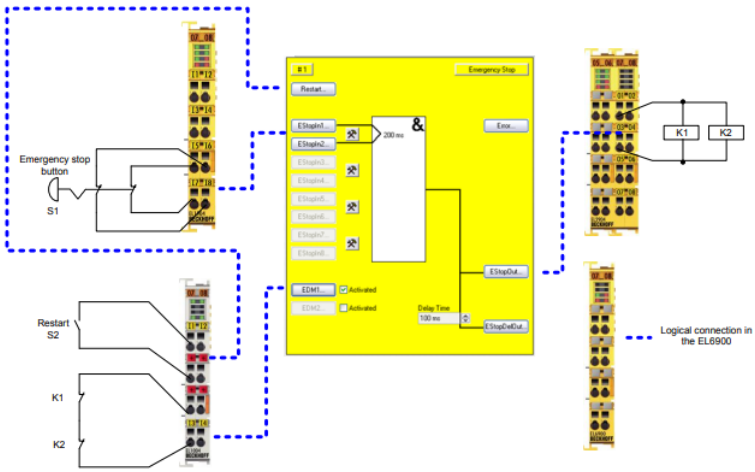

Figure 13. The EL1904 safety-card for ESTOP.

2.11 OMAC PackML

Often in a factory there are several of machines involved. These machines could be (and often are) from different suppliers which means that they most likely are using different program structures. However, when controlling a factory it is preferable to have some kind of overlaying structure so that the communication between the machines is simplified. This is exactly the purpose of OMAC PackML.

2.11.1 OMAC Standard

The OMAC standard was created to give a unified guideline how the layout of machines' operations and included units should be organized. This aims to simplify integration of new machines' into industrial applications for machine suppliers. For the "End User" this unification will enable integration with supervisory control systems. Mainly, because all interfaces are coherent and units of data types are similar for the interfaces [OMAC].

2.11.2 Overall layout

A typical PackML machine has three levels of hierarchy in this descending order: Machine module, Equipment module and Control module. To break down the code-structure in these parts makes it easier if a project has several programmers to coordinate and monitor the software (see Figure 14).

2.11.3 Machine Module level

The Machine level is the top layer of the PackML interface and what controls all other components. Although, the Machine is not just the primary program, it is also either HMI (Human Machine Interface) or OI (Operator Interface). HMI and OI is the visualized graphics on a physical screen that connects the machine operator to the machine (described later in the report). Furthermore,

the machine level is what actually runs the machine. This level takes input from the machine operator and gives instructions to the Equipment modules and deals with alarm or fault messages from Equipment modules.

2.11.4 Equipment Module level

The secondary level in the PackML interface and perhaps the most meaningful one. The sole purpose of an Equipment module is to perform an instructed function completely. An Equipment module directs the work to the Control module to complete a function. Additionally, there can be an unlimited amount of Equipment modules.

2.11.5 Control Module level

The lowest level in the PackML interface and the separate devices which performs tasks like motion of a motor. However, they can not complete a function without support of an Equipment module which instructs them. For instance, an axis block where several Equipment modules have axes, but they use each axis for different things. Although, the Equipment modules use the same Control modules for the motion along the axis [Beckhoff_Framework].

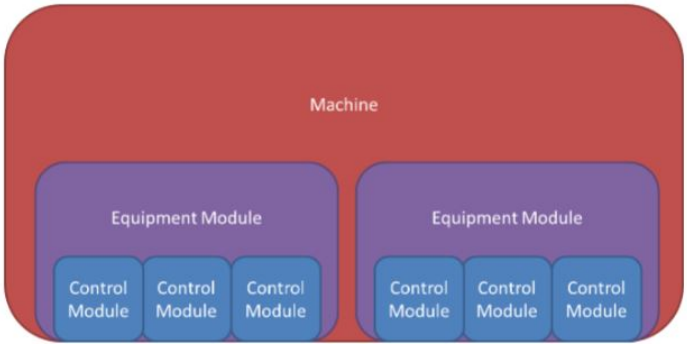

Figure 14. Code Hierarchy.

2.11.6 PackML State Operation

All machines with PackML configuration need modes and states to operate in. Assigning states/modes to different functionality is a must to make the code understandable and consistent while finding flaws becomes simpler too. Furthermore, it will prohibit so called "unintended consequences". Meaning that you should distinguish states/modes so they will not affect any other states/modes if something goes wrong. The optimal result for a machine is to always stay in the "Execute" state the whole time. In the "Execute" state the machine operates or is ready to take new instructions every time [OMAC].

2.11.7 PackML Control Commands

To go from one state to another state in the PackML state model, a command signal has to be sent. These commands are defined as ST_PMLc - Structure of a PackML command. To keep track of the status between the layers and

command signals the ST_PMLs - Structure of a PackML status was defined. The procedure of the command in the unit/machine Control System may be combined with public or private machine conditions. Public means change of global variables that can be reached anywhere in the code structure. Private means change of local variables that can be reached within one single location in the code structure - a map for instance. However, the unit/machine always needs to listen for incoming commands, even if the machine code will not make a state movement.

There are in total 10 state commands that can contribute to a change of PackML state, but only 8 will be mentioned here. From figure 14, all 17 PackML states and the corresponding state commands can be seen.

The first state command is Start, it moves the machine from an Idle to a Starting state.

Next one is the Hold command. This command should be executed when an internal error occurs in the source code or if a new job is expected during Production or Maintenance mode. It moves the machine from Execute to Hold state. Eventually the machine will be moved from a Hold state into an Unholding state.

From Unholding the machine goes to Execute by an Unhold command.

The State Complete (SC) command is an inner command that moves the machine to the next state and is used everywhere in the PackML interface model.

The Stop command will move the machine towards a Stopping state where it is halted. The Stopping command can be called from everywhere, except the light grey zones and white zones seen in Figure 15.

The Reset command can be commanded from the Machine operators HMI or decided to be automatic. Once a Reset command is sent, the state transfer is between Resetting and Idle state.

The Abort command can be sent automatically when an E-stop button is integrated for the machine. Otherwise, its called manual by the machine operator. This command can be called from any state in the PackML state model. The reached states will be Aborting where the machine is first halted and then power is shut off.

The Clearing command can then be used after the Aborting state has reached Aborted state. This command will make the Clearing state reset faults that occurred in previous state and power on the machine again [OMAC].

2.11.8 Production Order

For starting a production process, the Start signal must be triggered from an Idle state. When SC (State Complete) is achieved from Starting state, the Execute state is reached. Here the machine is operating until a Hold command a Stop command or an Abort command appear. When a Hold happens it goes into Hold and halts followed by going to Held. Here it waits until an Unhold signal is given and then from the Unholding state a SC will make the machine go into Execute. Once a production order is completed a Stop command will be generated which makes the machine go to Stopping state and halt here. Afterwards it goes by a SC to Stopped and wait for a Reset signal. Through a SC signal the machine is back in Idle state once again and everything repeats. When an Abort command appears the machine immediately halts and goes to Aborting state.

Once finished a SC makes the machine go to Aborted. From there a machine operator has to press Clear command to reach Clearing state. Once faults are fixed and power is on the machine reach Stopped state through a SC and awaits a Reset command. Whenever the Reset command happen the machine enters Resetting and moves the machine into its initial positions. Through a SC the next state will be Idle and everything repeats [Beckhoff_Framework] (see Figure 15).

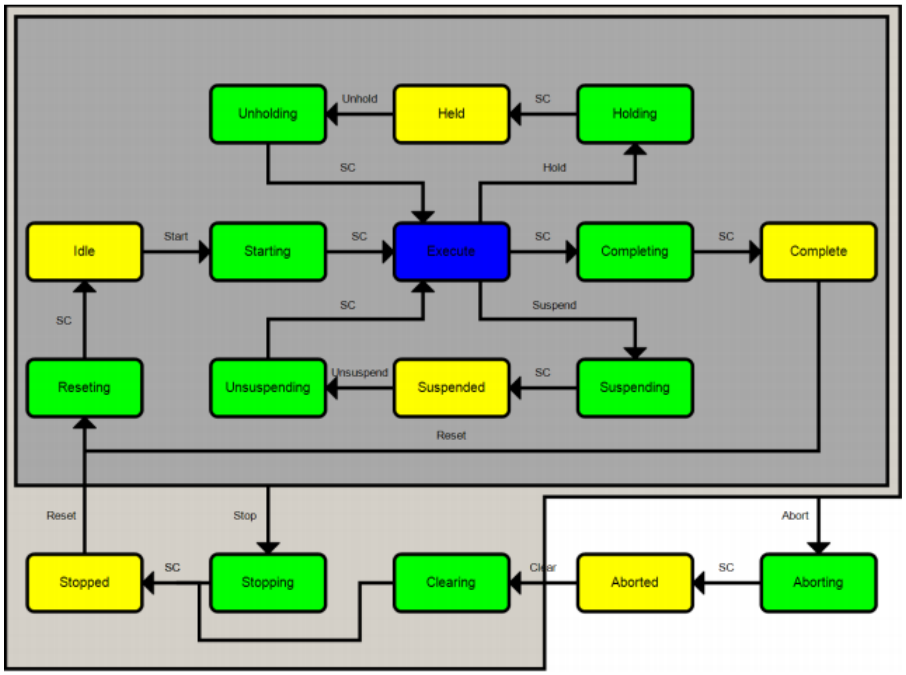

Figure 15. PackML State Model of Production mode.

2.11.9 PackML Modes

There are three different types of PackML modes in the OMAC standard. The top one is Production mode, the middle one is Maintenance Mode and the bottom one is Manual Mode. These modes can be switched between while the machine works in the Aborted, Stopped or the Idle state (see Figure 16).

The Production (Automatic) mode is based upon a normal automatic work sequence for an industrial process and includes total functionality of the PackML states.

The Maintenance (Step-in) mode generally has the same functionality as Production mode but every section or separate sections of the machines can be tested. This could be useful for testing code structure and error detection.

The Manual mode can be used for manual operation to test simple functions of the machine like moving, stopping or resetting [Yaskawa].

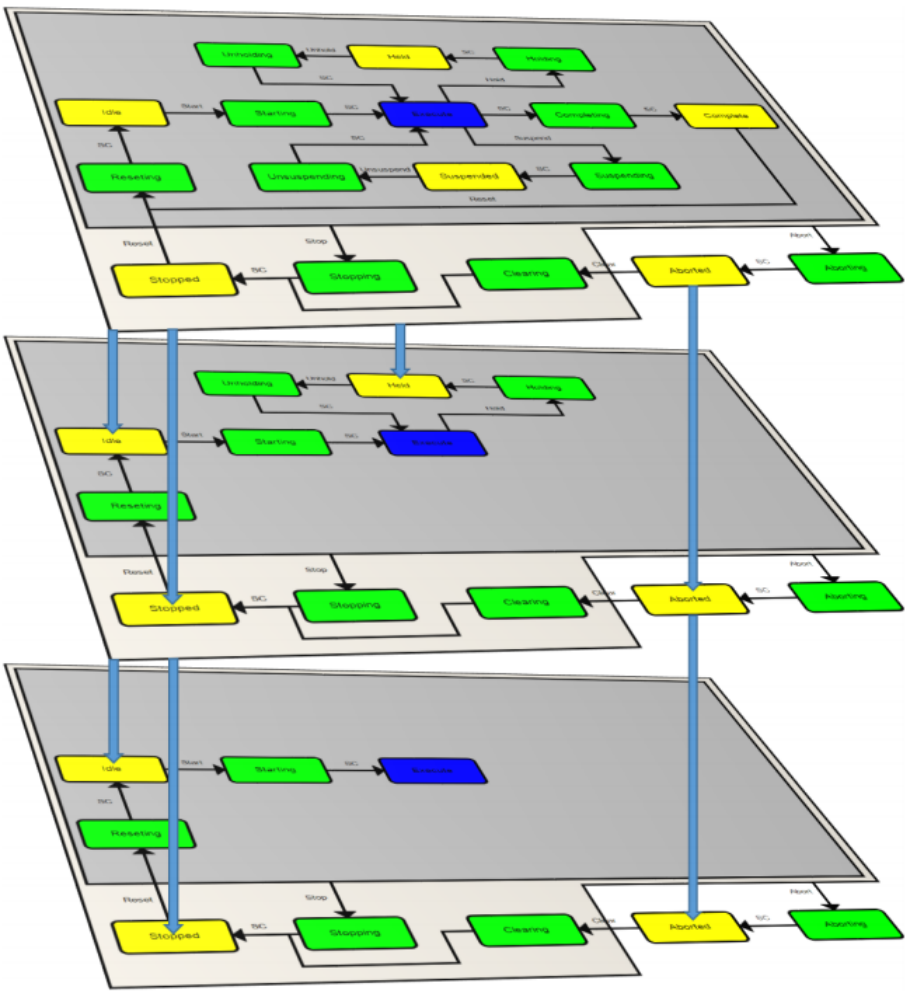

Figure 16. PackML State Model of Production mode (top), Maintenance mode (middle) and Manual mode (bottom).

2.12 Alarm

To have a functional alarm system can inform the machine operator before an error occurs or how to fastest deal with the current error. For production in a factory the uptime is essential. The uptime means the amount of time the factory actually produces goods or so called commissions [ControlEngineering_Alarm].

For purpose of an efficient alarm handling in the TwinCAT there exists a program called "Event Logger". Its a database that "logs "events", which means stores and reacts to alarms of different kind. Furthermore, in the TwinCAT library there exists an alarm struct of PackML - ST_Alarm. A new function block which combines functionality of the "Event Logger" and ST_Alarm can create a functional alarm handling [Beckhoff_EventLogger] .

2.12.1 Event Logger

The TwinCAT Event Logger takes inputs from "Event Classes" and dissembles them into two groups of events, "Messages" or "Alarms". "Event classes" are sets of "Events" blended together. The Event Logger efficiently communicates with the TwinCAT HMI, the TwinCAT XAE (runtime) and the TwinCAT XAR (environment) by exchanging data between the programs (see Figure 17).

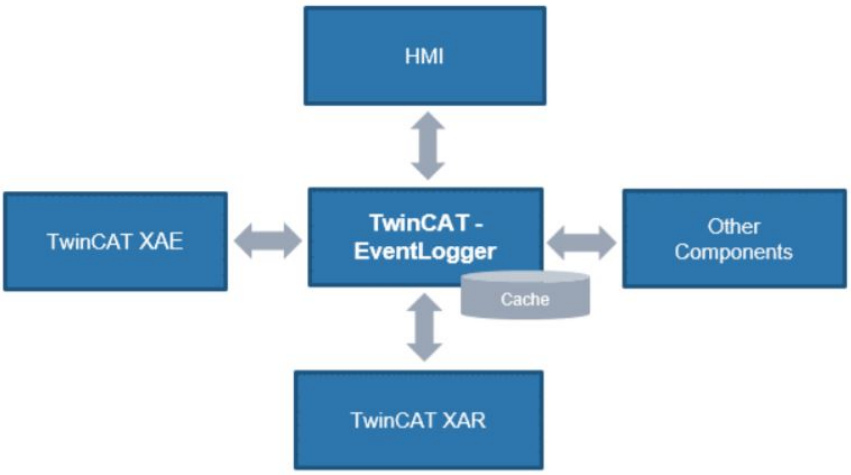

Figure 17. TwinCAT EventLogger.

Each Event has an Event-ID making it unique within an "event class". Every event has a "Text" string, information about the Event. There is also a "Source Info", information about where and how the event appeared. Furthermore, there exist a JSON attribute String to each "Event" [Beckhoff_EventLogger]. The JSON (JavaScript Object Notation) means a small and understandable text for organising data used in XML code [SquareSpace].

An "event class" can be created from a "TMC-editor" program from the "Type System" in the PLC. Each "event class" can also be edited and opened. Additional "event classes" that could be created are ADS return data and other system related ones (see Figure 18).

When creating a new PLC-project a global variable GVL_TC_Events is created inside the PLC. TC_Events contains "Event Classes" as input to it and refreshes after saves in the "TMC editor". In the TC_Events the inputs are, "event-ID", "Severity class" and "UUID". The "Severity class" can be set by the programmer as a number, 0 to 6 and the higher the number the more critical it is. For the scope of this thesis only the "Severity classes" "Warning" and "Error", which corresponds to the values 4 and 5 are used. The "UUID" (Universally Unique Identifier) is a number that defines a unique event [Beckhoff_EventLogger].

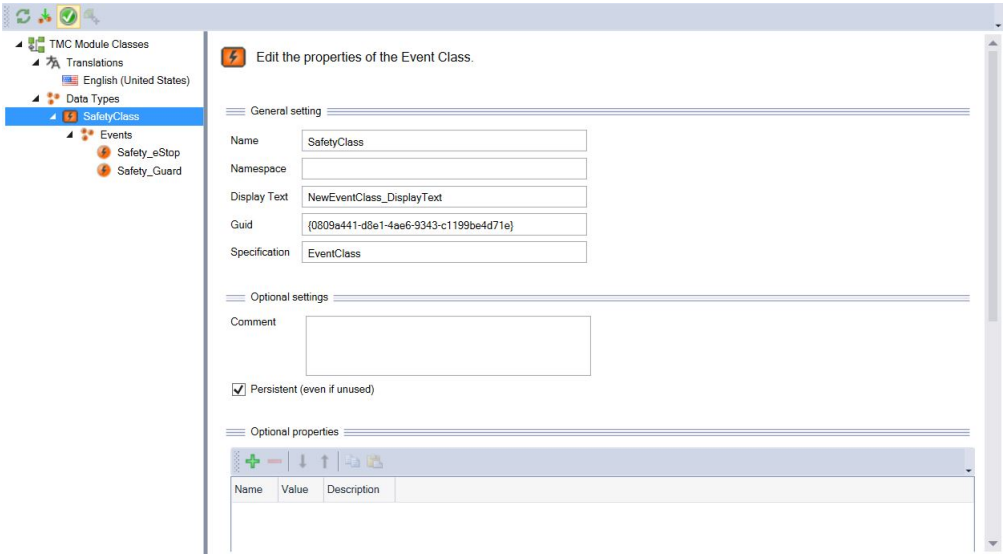

Figure 18. TMC-editor for TwinCAT EventLogger.

2.12.2 PackML Alarm

As previously mentioned, the PackML alarm handling uses a struct called ST_Alarm. This struct constitutes of a "Trigger signal", an "Id" number, a "Value", an error "Message", an error "Category" and two arrays storing up to six time occasions each called "DateTime" and "AckDateTime". "DateTime" means when an error has happen and "AckDateTime" when an operator read the error message [Beckhoff_Alarm].

2.13 Drive systems

A typical drive could be a speed drive system which is supposed to manage the speed and torque of a motor. One way to tune such a motor is by adjusting physical loads so the drive responds to varying loads.

Drives have different applications within mathematical computations and could be advanced to implement. Some common features could be to use gates with boolean logic (AND, OR etc.) and PID regulators with control data. Another feature of a drive system is that it enables a simple integration between the PLC and its HMI since all data exist within the PLC.

To program drive systems through PLC code could be challenging, both in terms of difficulty and time. Generally, control data for the drive systems can be passed on between different manufacturers' software, which reduces the number of programs an engineer needs to use. An example of this could be if Beckhoff would implement control data from Siemens [ControlEngineering_Drives].

2.14 Human Machine Interface (HMI)

When the term "Human Machine Interface" (HMI) first was used it described all interfaces between man and machine. Today it is most commonly used in

industrial- and automation applications. It is one of the most important interfaces the operators are working with. The HMI is created using software and the application can run on any computer/phone/tablet or physical panels (in many cases) [Cenito]. While HMI is the most common term for this technology, some other terms could be mentioned: Operator Interface Terminal (OIT), Operator Terminal (OT), Man-Machine Interface (MMI) or Local Operator Interface (LOI). It is also not unusual to compare HMI to Graphical User Interface (GUI). They are similar but not synonymous. GUIs are often leveraged within HMIs for visualization capabilities [InductiveAutomation].

2.14.1 TwinCAT HMI

TwinCAT HMI is a tool for creating HMIs. Its integrated directly into Visual Studio (like TwinCAT). For the configuration a graphical WYSIWYG (What You See Is What You Get) editor is used, which makes it usable without programming. From the toolbox (see Figure 19) controls can be arranged on the UI and mapped with real-time variables (e.g. from the PLC). User controls is used to create and configure specific controls. It is also possible to develop own controls by using JavaScript. The functionality can be tested from the real-time system with live data as the development is ongoing. During this "live mode" modifications are also possible.

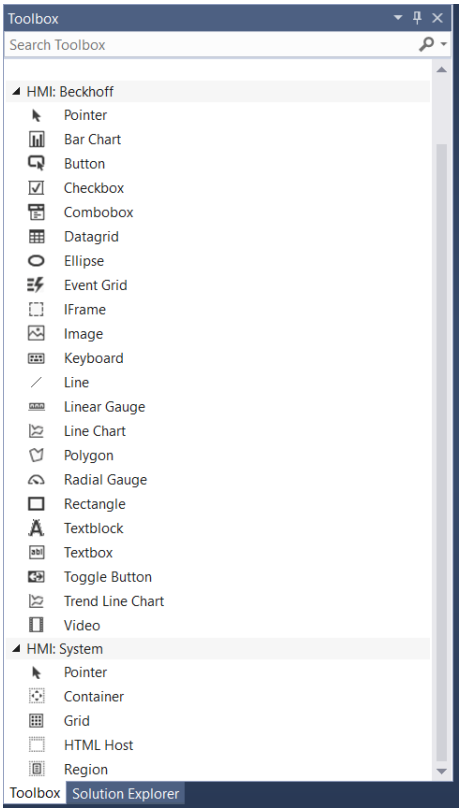

Figure 19. Toolbox from the TwinCAT HMI.

The logic of the HMI can be implemented either as a server extension or on

the client side in JavaScript. Furthermore the server extension allows to develop extensions in C#.

The resolution and orientation of the HMI will adapt automatically. This is because the HMI is responsive and web-based (based on HTML5 and JavaScript). Thus it is possible for the same page to be used for different display aspects, ratios, sizes and orientations.

The following section will explain the tools used from the HMI toolbox in this project [Beckhoff_HMI].

Desktop

The start page of the application is called "desktop view". This is the base platform to create the visualization. To increase the tidiness the visualization of the desktop is often split into various parts. This is done by creating various content objects. These content objects can then be displayed at desired places on the desktop. A common method is to firstly divide the desktop into regions on which the content will be displayed.

Content

The content (content control) is an independent container object that can be visualized in a region. The content is used as a base for the creation of all sorts of tools, such as buttons, textblocks etc. It is also possible to create regions within a content. This makes it easy to get a good structure of the program when it comes to handling several "pages".

Region

A important tool is the regions. It is a container for content controls. By using a special attribute it is possible to define the "Target Content", i.e. choose which content to be displayed in the region.

Button

The button provides visual feedback when the background is switched. In other words, it is a simple switching element. With the button comes various "operator events", which allows the button to be programmed to do certain things as the event is fired. These events can be actions such as "OnPressed" (executed as the button is pressed) or "onStatePressed" (executed as long as the button is pressed).

Toggle Button

Another type of button is the "Toggle Button". Like the regular button its a switching element. What differs is that it can switch between two states. With the "Toggle Group" attribute it is possible for several toggle buttons to interact with each other. An example of this is that if one toggle button is already activated, by activating another button it would be deactivated.

Textblock

A "Textblock" is simply a control in which a text can be displayed.

Event Grid Control

The "Event Grid Control" is used to display alarms and messages. It is a tabular that automatically displays the alarms and messages of the target system. It is possible to confirm alarms directly in the control.

Changing the language

In TwinCAT HMI there are localized text defined by key/value pairs. It is possible to add several translations for each key. In the localization editor each language can be edited. Where it is desired to use several languages these are simply defined in the localization file and called upon by the key when using it in the HMI.

2.15 Recipe

Recipes that are used in production processes are simply stores of data, which are specified to a particular product. This data can be called upon and then implemented to produce the product. The recipe contains a set of parameters important to the production process .

The area of application for recipes is only when multiple products are being produced. In the case of a single product, the parameters for the production are stored in the automation system and repeated time after time. On the other hand, when multiple products are being made and the parameters among them vary, recipes make it possible to quickly and precisely change from one product to the next. The alternative would be to have an operator change the parameters manually each time a change of product is needed [Automationdirect].

2.15.1 TwinCAT Recipe Management

The recipe management in TwinCAT enables the activation of symbols, which are managed in recipes. Online values from the development environment as well as from the visualization clients can be saved by the target system in a recipe.

Recipe types and recipes

There is a fundamental distinction between recipe types and recipes. A recipe type is a general description of a set of symbols. With the recipe type as a base it is possible to create various recipes. Values for the symbols, which are defined in the recipe type, can be saved in a recipe.

Recipe Management Sample

Beckhoff provides a recipe management example which serves as the point of entry to the recipe management and can be extended as desired. The example consists of a TwinCAT HMI project, which can be copied and used for own practice (see Figure 20).

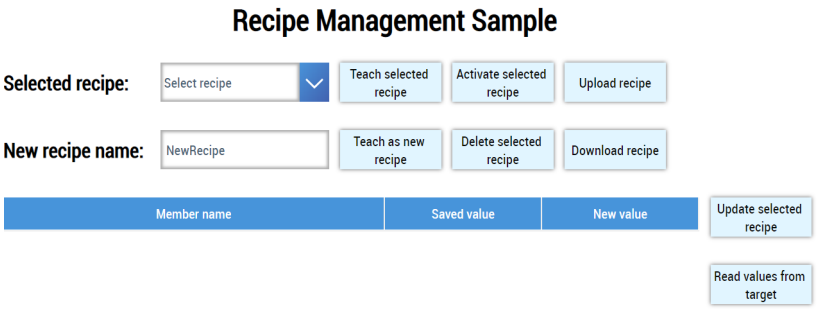

Figure 20. The Recipe Management Sample from Beckhoff.

The functions of the recipe management are described below.

Recipe selection: The available recipe types are displayed in the combo box. If there exists a recipe inside a folder, the relative path to the recipe is displayed. The purpose of the combo box is to handle the selection of the recipes that is used as a basis for the other fields an buttons in this example-program.

Teach and activate recipe:

- Teach selected recipe - Currently selected recipe is taught again which means that the current online values are adopted from the PLC and saved in the recipe.
- Teach as new recipe - With the current online values from the PLC (on the basis of the currently selected recipe) a new recipe is created. For this to work a new name for the recipe must be entered in the text box.
- Activate selected recipe - The currently selected recipe is activated which means that the values that are stored in the recipe are written to the PLC.
- Delete selected recipe - The selected recipe is deleted.

Display and edit recipe: Enables the display and editing of the recipe symbols (members) of the selected recipe.

- Datagrid - Currently selected recipe is loaded and the members are displayed in the data grid. The column "New value" has two functions. Firstly, it is used to receive new values from the members of the recipe, which are saved by using the button "Update selected recipe". Secondly, it is used to display the current online values of the members for the purpose of comparison with the stored values. This is done by clicking on the button "Read values from target".
- Update selected recipe - Saves recipe with the new values from "New value".
- Read values from target - Reads out the current online values of the recipe members and displays them in "New value".

3 Method

3.1 Training

The first approach of achieving fundamental knowledge of TwinCAT was to take courses in TwinCAT Basic and TwinCAT Motion. TwinCAT Basic focused on giving an overall view of coding, libraries, documentation, functions and different types of PLC code. TwinCAT Motion focused on how to run both virtual and real motors on an axis for machines and even synchronize motors together.

3.2 Acceptance test criterias

To clarify what actually needs to be achieved some acceptance test criterions' were defined. The following points show which requirements the machine should fulfill.

- Simulate the function of the machine;
- The machine should operate in VC environment;
- The 12 axes should be coupled to three virtual axes in the PLC;
- Implementation using the PackML-structure;
- Alarm handling;
- Safety:
  - Emergency stop;
  - Open protective gates around the machine;
  - Jog the machine with constant contact control;
- EtherCAT simulation;
- Simple HMI:
  - 3D-coordinates for the machines axis should be visualized in the HMI;
  - Start/stop/jog the machine through buttons;
  - Recipe function;
  - Error acknowledgement button;

3.3 Learning the basics of the function blocks

The following sections describe the approach of learning the basic functionality of the most common motion blocks in the TwinCAT environment.

MC_Power

The first step was to create an axis. To do this the library TC2_motion had to be imported. This was followed by making an axis called "AxisRef" with reference to AXIS_REF. To turn the power on the axis the MC_Power block

was used. To use the block the input "Axis" was set to AxisRef followed by setting "Enable", "Enable_Positive" and "Enable_Negative" to true. When the output "Status" was true the axis was turned on and able to take further instructions.

MC_MoveAbsolute

The second function block that was tested was the MC_MoveAbsolute. This was initiated by setting the inputs: "Axis" to AxisRef, "Position", "Velocity, "Acceleration", "Deceleration" and "Jerk" to desired values. The motion was then started by setting "Execute" to true. When the output flag "Done" became true this indicated that the motion had completed.

MC_Reset

Furthermore the function block MC_Reset was tested. This was simply done by using the MC_Reset block on the AxisRef and setting the "Execute" to true. When the output "Done" was true the function block had succeeded.

MC_Halt

While the axis was in motion and supposed to brake in a controlled fashion to zero speed the function MC_Halt was executed. To do this the inputs "Deceleration" and "Jerk" was set to desired values. MC_Halt was then called upon the AxisRef followed by setting the input "Execute" to true which would halt the axis. As for the other blocks when the flag "Done" became true it was an indication that the block was done and in this case the motion had halted.

MC_Stop

While the axis was in motion and supposed to stop immediately the function block MC_Stop was executed. To do this the inputs "Deceleration" and "Jerk" was set to desired values. MC_Stop was then called upon on the AxisRef followed by setting the input "Execute" to true which would stop the axis. Like for the halt case, when the flag "Done" became true it was an indication that the block was done and the motion had been stopped.

MC_GearIn

The next step was to create several axes and try to couple them. To create more axes the same procedure as before was used. When two axes (AxisRef and AxisRef2) were available the function block MC_GearIn was tested. This was done by setting the inputs: "Master" to AxisRef, "Slave" to AxisRef2, "RatioNumerator" and "RatioDenominator" to 1, followed by setting "Acceleration", "Deceleration" and "Jerk" to reasonable values. When setting the "Execute" to true the block would begin. As the flag "InGear" became true the coupling was done.

MC_GearOut

To decouple a slave axis from its master the MC_GearOut was called upon. This was done by setting input: "Slave" to AxisRef. When setting the "Execute" to true the block would begin. As the flag "OutGear" became true the decoupling was done.

3.4 Industrial transportation machine in Visual Components

When receiving the model of the 12-axis machine it was time to start separating the axes of the machine. Since the model was only a step-file (CAD-file) there were no dynamics predefined. To separate the axes a specific tool in Visual Components was used, the modelling tool. This tool made it possible to choose in detail (down to the very last screw) which components that were going to operate on which axis. As the robots were composed of four 3-axis modules this had to be performed on each one of them. Additionally, this modelling tool was used on the Y-axis lifters which the four robots were attached to. Once the whole model had its dynamics defined, all the axes were set to move along a straight path in either X,Y or Z -direction.

3.5 ADS-communication with Visual Component

When some basic knowledge about the PLC interface was gained, it was time to learn more about how ADS communication can be established between a TwinCAT PLC and Visual Components. By activating configurations with I/O:s and licenses as well as running TwinCAT in the PLC project, the PLC was ready to operate. In Visual Components, ADS mode was configured by enabling online mode as well as configuring a server for ADS. Once the server configuration was done on Visual Components, the ADS pathway could be established between Visual Components and TwinCAT. There were two options, "simulation to server" and "server to simulation". Through "simulation to server" Visual Components generated code which gave values in TwinCAT, for example a coordinate or a sensor signal. Correspondingly, "server to simulation" meant that TwinCAT generated code to Visual Components, for example motor signals or command signals to different axes.

3.6 Program structure

The next step was to figure out the Program structure of the PLC code - how everything should be arranged in relation to PackML, coding structure and the system structure.

OMAC PackML

The OMAC PackML model has in general cases 17 states, up to 18 if an "ESTOP" (electronic stop) state is included. Although, three of the states

regarding Suspending of the machine was decided not be included in this project. The reasoning was that information about the surrounding environment where the machine could operate was beyond the scope of this thesis. However, states regarding Holding was kept since internal faults could occur in the source-code. When internal faults occurs, the machine should be able to pause so a machine operator could go and fix the issue. Another thing that the holding states should handle is when the machine is starved (which very much could be the state in this project).

The "ESTOP" state got to be an external state outside the PackML structure. It was called from any state in the structure, and both stops and aborts any current process. The only way to return back to the machine states were that specified safety conditions had to be fulfilled by the machine. Based upon these specifications, 15 states of the PackML were decided to be implemented.

Coding Structure

In terms of coding structure of the machine; Control modules, Equipment module and Machine module were organized for the machines' different axes. Therefore, the specific functions of an axis, such as making it go up/down etc. would be implemented in a control module. The plan was to implement three control-modules for the machine, regarding X,Y and Z-axis direction. Furthermore, one Equipment module was used to simulate the four transportation robots to follow the same programmed route. Finally, one Machine module was implemented to call the Equipment module and handle the HMI of the machine. Examples of the HMI functions where start, stop, jog and "ESTOP", which were implemented for the machine operator.

System Structure

In terms of the system structure , one laptop, one PLC device and the 12-axis machine were chosen (see Figure 21, 22 and 23).

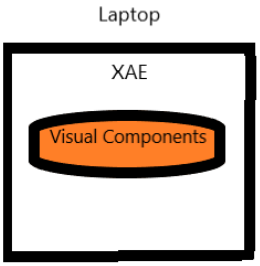

Figure 21. The system structure of the laptop.

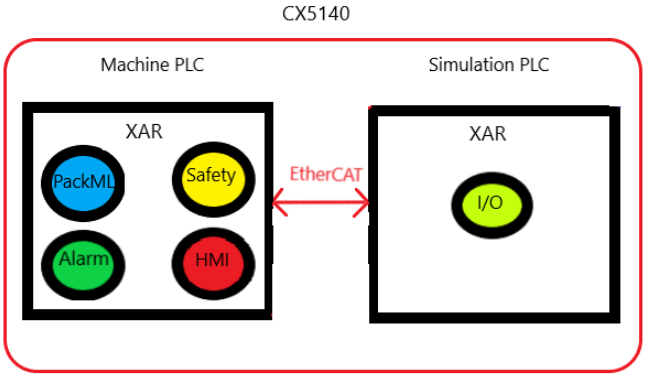

Figure 22. The system structure for the CX5140 with the Machine PLC-project and Simulation PLC-project.

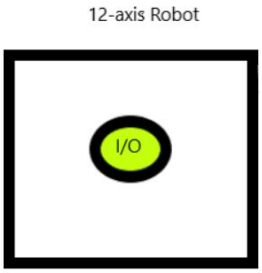

Figure 23. The system structure of the 12-axis machine.

The programming environment (XAE) was on the Laptop - where all code was written and also Visual Components to simulate the 12-axis machine (see Figure 21). All the data that was transferred between the Laptop and Machine PLC was done by ADS (see Section 3.5).

The CX5140 PLC device used two PLC projects, the Machine PLC-project and the Simulation PLC-project. In the Machine PLC-project, the code for

PackML modules, the EtherCAT safety, the Alarm Handling and the HMI were held. The Simulation PLC-project had code which was responsible for the EtherCat simulation. The EtherCAT simulation meant to handle a real-time fieldbus communication together with I/O:s. Both the Machine PLC-project and the Simulation PLC-project were sharing the same PLC run-time (XAR) since they were existing in the same PLC-solution. The Machine PLC-project and the Simulation PLC-project were also connected together by an EtherCAT cable connection through a CU2508 EtherCAT switch (see Figure 22).

The full system structure was then including the 12-axis machine to be able to replace the Simulation PLC-project (see Figure 23). Basically, disconnecting the Simulation PLC-project from the I/O:s and pluging the I/O:s back into the 12-axis machine. Thereafter all coding would be generated from the Machine PLC-project directly into the 12-axis machine and back. In theory the 12-axis machine should operate in reality the same way as the simulation would in Visual Components (see Figure 24).

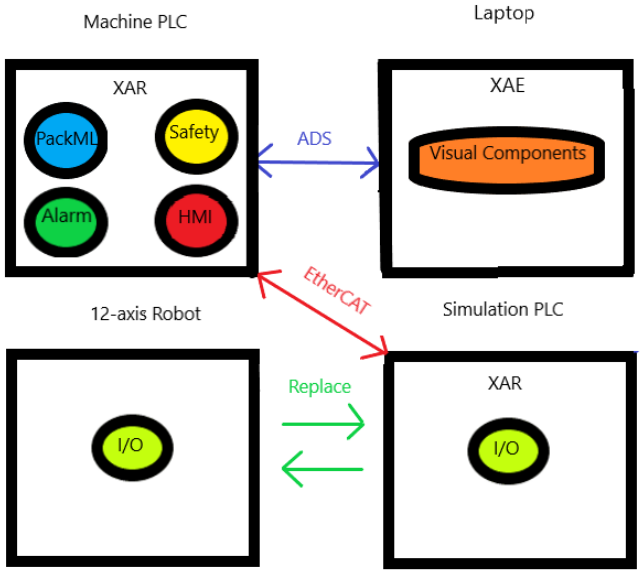

Figure 24. The full system structure of all communicating devices.

3.7 The MAIN-program

The MAIN-program was what actively ran the PLC code in both the Machine PLC and the Simulation PLC. The MAIN-program would then call upon the current Machine module state, which then calls the corresponding state in the Equipment module and then the Control module. As previously mentioned, the three Control modules could handle movement, stop, reset and coupling. Hence, running the MAIN-program, makes the 15 virtual axes run (12 slave axes and 3 master axes). Through declaration of position variables in the Equipment

Module, the coordinates of the 15 axes could be updated. The run-time (XAR) of TwinCAT would then yield an update frequency of the position variables every 10 ms.

3.8 PackML

As previously mentioned, the code of the PackML structure took the use of the run-time (XAR) of the Machine PLC (see Figure 25). To implement the PackML structure it was decided to be the first out of the four major code parts to be implemented.

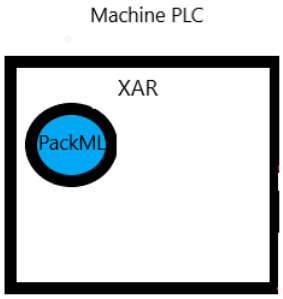

Figure 25. Inclusion of PackML in the system structure.

3.8.1 Implementation of the hierarchy (Machine, Equipment, Control)

The first procedure was to implement PackML maps for each of the three Control modules. The control modules would implement ten acting states and one execute state. The acting states worked sequential once a requested signal entered the state (see Figure 15).

The second procedure was to implement PackML maps for the single Equipment module. Here three folders were created, one for manual, one for production and lastly one for maintenance. In the manual folder, ten acting states were used for jogging and testing of the four transportation robots, either separately or together. Additionally, for both the production map and the maintenance map all ten acting states were implemented.

The last procedure was then to implement on PackML maps for the Machine module. Like in Equipment module, three maps were created to include all the modes. In the manual map, the states instructed the Equipment modules to use its manual states. Likewise, in the production map the Machine module instructed if the Equipment module should use its production states. In the same way the Machine module could then instruct the Equipment module to use the maintenance states.

3.8.2 Defining Machine-, Equipment- and Control modules

In this project the Machine module was chosen to be equivalent to the "factory level" or "production line level". This meant that the Machine module would control every machine in the factory or production line (which in this case only was the 12-axis machine).

The Equipment module was simply chosen to be the 12-axis machine.

Lastly the Control modules were chosen to handle simple tasks like power up an motor. Therefore three Control modules were defined for X, Y and Z.

3.8.3 Testing the behaviour of OMAC PackML

When the actual PackML base had been created with folders it was time to start running some code in the states to observe the behaviour of the structure. To begin with, this was done by simply enabling the axes in the aborting state at the Control module level. Since initially (when starting the program) the Aborting state was the state running, the axes were enabled as soon as the PLC went into Run-mode. Furthermore some code in the "Clearing" state was written that would reset the axes and move to some given position. For this code to run the machine had to be commanded with a Clear-command, which was done on the machine level by setting the "MachinePMLCommand" to Clear. A lot of similar tests was executed to get a better understanding of the behaviour of the structure.

To send information about what code that ran on a lower level module to a higher level module an interface of type I_PTP was created at each PackML level. The interface I_PTP used MC2_functions which controls point-to-point movement. When an Equipment module called upon the values of a Control module, it had to create I_PTP instances to map the Control module I_PTP outputs to it. One I_PTP instance had to be declared for each of the 12 virtual axes for this to work. Within each instance, the 12 virtual axes were connected to one of the three corresponding master axes.

3.8.4 Machine level

In this section the functionality of each state is described for the Machine module.

Idle

The "Idle" state was only implemented for the Manual mode. Here the machine operator made a decision which axes that should be jogged right now. Once done with the decision, an automatic "Start" command would be sent and eventually the "Starting" state got reached.

Held

The "Held" state was only implemented for the Production and the Maintenance mode. Here a boolean signal "bSensor" simulated when a conveyor

had a product ready to be picked up by the four transportation robots. While "bSensor" equaled false, the current mode would remain in the "Held" state. Once "bSensor" equaled true an "Unhold" command was sent and eventually the "Unholding" and then finally the "Execute" state would be reached.

3.8.5 Equipment level

The following section describes the functionality of the states in the Equipment module.

Starting

In the Starting_Production, the starting values were set for MC_MoveAbsolute. Once finished, a SC command would be sent and the "Execute_Production" state eventually reached.

In the Starting_Manual, the starting values of the MC2_Jog and MC_GearIn were implemented. Furthermore, the coupling was activated for the chosen axes in "Idle" state. When all the chosen axes were coupled a SC command would be sent and the "Execute_Manual" state eventually reached.

In the Starting_Maintenance, the starting values were set for MC_MoveAbsolute and MC_Halt movement. Once finished, a SC command would be sent and the "Execute_Maintenance" state eventually reached.

Execute

In the "Execute" production sequence the machine started of in a "Homing" position and moved to a "Waiting" position in X,Y and Z. Then when a "bSensor" signal gave true from a conveyor (a product ready), the 12-axis machine went to a "Grab" position in X,Y and Z to pick up an object. Once finished, the 12-axis machine went to a "Destination" position in X. Thereafter, the axes were lowered to the "Drop" position in Y and Z direction to drop of the product. Additionaly, all the axes of the 12-axis machine went back to the "Wait" position to repeat the same procedure again. The procedure would start of immediately if "bSensor" gave true. When "bSensor" gave false the 12-axis machine stayed in "Waiting" position.

For Execute_Manual the execute process was based upon jogging of the machines, separately or together. The axes which were coupled from the "Starting" state could now be moved in X,Y or Z direction.

The Execute_Maintenance sequence was the same as for Execute_Production while the "Step Forward" button was toggled. However, when the "Home" button was toggled the master axes in X and Y position moved back to a homing position.

3.8.6 Control level

Here the functionalities of each state at the Control module is described.

Aborting

The first operating state to be implemented was "Aborting". Through an Abort command from every possible state, the 12-axis machine were halted and their power turned off. Once halted and power turned off, a SC command moved the current state to "Aborted".

Clearing

Next operating state to be implemented was the "Clearing" state. By a Clear command from the Aborted state, all errors and axes couplings were reset. Once all axes were reset their power was put "on" again. When these procedures were finished, a SC command moved the current state to "Cleared".

Resetting

The Resetting state had three different implementations. One for Production, Manual and Maintenance mode.

In Resetting_Production, all axes were moved to their "Home" position in X,Y and Z Control module. Thereafter all "slave" axes were coupled to their corresponding "master axis" in X,Y and Z Control module. When this was finished for each Control module, they sent a SC command to go to "Idle" state.

In Resetting_Manual, the Resetting state was not implemented and only sent a SC to reach "Idle" state.

In Resetting_Maintenance, all axes were moved to their home positions in X,Y and Z Control module when a "Homing" button was toggled. Once all the home positions were reached, the "Home" button got deactivated. Thereafter all "slave" axes were coupled to their corresponding "master axis" in X,Y and Z Control module. When this was finished for each Control module, they sent a SC command to go to "Idle" state.

Stopping

The Stopping state had the same implementation for Production and Maintenance and a different one Manual mode.

In Stopping_Production, the Stopping state was not implemented and only sent a SC to reach "Stopped" state.

In Stopping_Manual, at first all three master axes in X,Y and Z were halted. Once halted, they were all decoupled. When all the axes were decoupled, the "Stopping" state sent a SC to reach the "Stopped" state.

In Stopping_Maintenance, the Stopping state was not implemented and only sent a SC to reach "Stopped" state.

Holding

In the "Holding" state all master axes in X,Y and Z axes were halted. Once all master axes were halted a SC was sent and the "Held" state eventually got reached.

Unholding

In the "Unholding" state only a SC was generated and eventually the "Execute" state got reached.

3.9 EtherCAT Safety

The next major coding step was to implement the EtherCAT safety. Firstly, the 12-axis machine needed an "ESTOP" functionality which was planned to be replaced by a physical emergency stop button. This emergency stop button could be pressed down whenever by the machine operator and should both stop and turn off the 12-axis machine. Secondly, it was investigated the possibility of using a "Safety-gate" on an outer safety environment around the 12-axis machine. The "Safety-gate" itself should stop and turn of the 12-axis machine when opened, otherwise do nothing. The outer safety environment was supposed to be constructed in Visual Components around the 12-axis machine (see Figure 26).

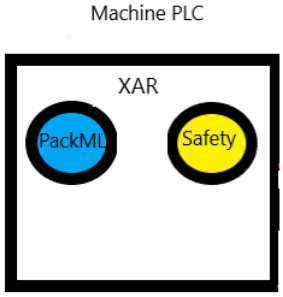

Figure 26. Inclusion of EtherCat safety in the system structure.

ESTOP Safety

When the three main Execute modes had been implemented it was time to investigate EtherCAT security. The first step was to physically connect an EtherCAT safety card to the PLC - an EL6910 with TwinSafe I/O:s. Once connected physically a safety project could be created within the PLC program. This safety project would check if any PLC data got corrupted or a fault appeared on the virtual axes for the PLC.

Next step was to bind the the "ESTOP" button to an abort command on the safety card so the machine could stop whenever a security breach occurred. To start off, two boolean global variables were mapped as inputs to the PLC from a Safety card to check the safety in the PLC. The variables were called "bSafetyOk" and "bSafetyOkDel" that always were true, unless an error appeared which turned them false. Then it was written two additional boolean global variables that were mapped as outputs - from PLC to safety card. They were called "bErrorAck" and "bRestart". "bErrorAck" equaled true when a machine operator would be aware of an error and toggle it on a HMI or false otherwise. "bRestart" would need to be set true from a machine operator whenever an "ESTOP" command would require a new trigger, otherwise it could be false.

Furthermore, to test the "ESTOP" button the plan was to set an entrance condition outside all the code in the Machine states for the three different modes. The condition was to press the "ESTOP" button which would make "bSafetyOk" turn false. By an if statement checking when "bSafetyOK" equaled false, a "MachinePMLCommand" was generated that sent an "Abort" command to all levels of the hierarchy. Then the Machine module, Equipment module and the Control module would enter the Aborting state. Once the "Abort" Command happened "bErrorAck" would turn false as well as for "bRestart". Then the machine operator manually has to set these variables true through the HMI and send a "Clear" Command. While the "ESTOP" button was reset and not pushed down "bSafetyOk" turned true, and all the PLC code would run normaly (see Figure 27).

Once the condition was implemented, the next step was to test the "ESTOP" in the Execute_Production, Execute_Manual and Execute_Maintenance states.

Here the "ESTOP" button was pushed down while the machines were running and they would immediately halt and go from Execute to Aborted state. Once the four transportation robots were decoupled and all the faults reset it could go through all states back to Execute.

The next step was to make "ESTOP" work in the Resetting_Production and Resetting_Maintenance states. Here the "ESTOP" button was pressed down while a "Homing" sequence. Then the four transportation robots would immediately go from Resetting to Aborted state. Like in the case of execute, the four transportation robots were decoupled and had all their faults reset and could eventually go back to Reset

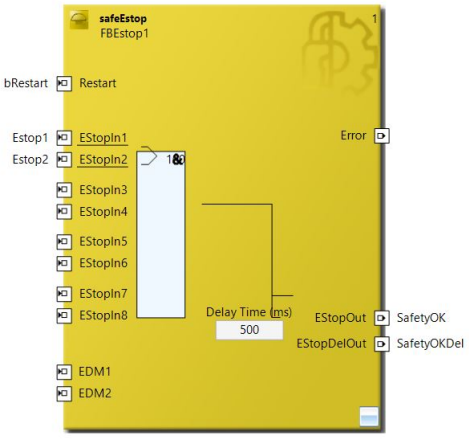

Figure 27. TwinSafe I/O "ESTOP" card.

Guard safety

After the implementation of eStop safety, the next safety card to implement was Guard safety. A copy of the same EtherCAT safety card was used for this occasion - an EL6910 with TwinSafe I/O:s. However, the input and output signals had other purposes. This safety card was added to the current safety project within the PLC program. Then it would check if a pair of "magnetic sensors" would loose contact, and then cut the current to the PLC. This was simulated by the use of a boolean PLC variable "bOpenDoor", generated from the HMI. This was visualized in VC (Visual Components) as a door open at 90 degrees. The door was located at a safety zone around the four industrial transportation robots. When "bOpenDoor" became true it meant that the "magnetic sensors" lost contact. On the contrary when "bOpenDoor turned false the "magnetic sensors" had contact again.

Furthermore, two boolean global variables were inputs to the PLC from a "Guard safety". These variables were "eStopIn1Guard" and "eStop2Guard" supposed to symbolize two "magnetic sensors". By using two input pins on the EtherCAT "Guard safety" card the "eStopIn1Guard" and "eStopIn2Guard" could be realised. When the two parts of the "magnetic sensor" would be kept away from each other ("bOpenDoor" equal true), the "eStopIn1Guard" and

"eStopIn2Guard" gave true signals. On the contrary when they were in contact ("bOpenDoor" equal false), both "eStopIn1Guard" and "eStop2Guard" gave a false signal. To send an output to the PLC from the "Guard safety" card a new boolean global variable called "bGuardOk" had to be created. "bGuardOk" equaled true when "eStopIn1Guard" and "eStopIn2Guard" were false - meaning safety alright. On the opposite, "bGuardOk" equaled false when "eStopIn1Guard" and "eStop2Guard" were true - meaning safety not alright. Another boolean global variable was declared as output called "bRestart2". "bRestart2" was set to the value of "bRestart" since they were supposed to work on the same way. "bErrorAck" was now mapped to both the "eStop safety" card and the "Guard safety" card. Like in the case of "eStop Safety", "bErrorAck" equaled true when a machine operator is aware of an error and toggle it on the HMI (see Figure 28).

Additionally, to test a the "magnetic sensor" the plan was to add an condition to the if statement used for the "eStop Safety". The new condition would then be if "bSafetyOk" equal false or "bGuardOk" equal false. Once these conditions go to false a "MachinePMLCommand" was generated that sent an "Abort" command to all levels of the hierarchy. Then the Machine module, Equipment module and the Control module would enter the Aborting state. Once the Abort command happened "bErrorAck" would turn false as well as for"bRestart2". Then the machine operator manually had to set these variables true through the HMI and send a Clear command.

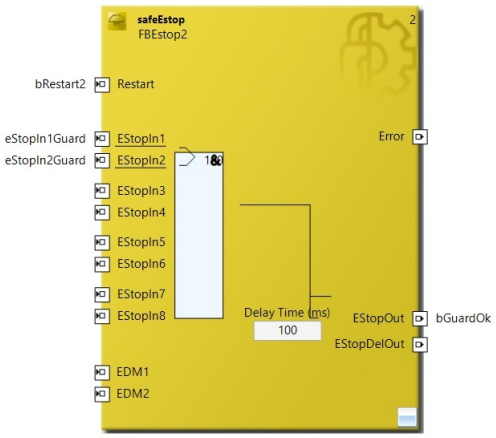

Figure 28. TwinSafe I/O Guard card.

3.10 Alarm handling

To inform the machine operator when a problem was about to appear inside the machine an alarm handling was required. The two different types of alarms that were decided to be implemented were "Warning" and "Error". A "Warning" was needed since it could inform that a fault was about to happen and an "Error" when a fault had already happened. Consequently, two methods had to be created in the Machine PLC - M_AlarmInstances and the other one

M_Safety. Both of these methods were declared and implemented in the Machine module and the Control Module. For the Equipment module only the M_AlarmInstances was needed to be implemented (see Figure 29).

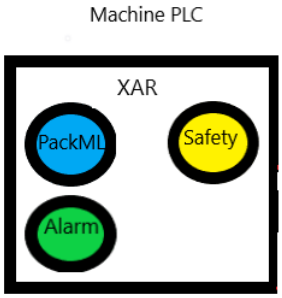

Figure 29. Inclusion of Alarm handling in the system structure.

To create a specific alarm for one "event" each one of them needed the function-block type fb_AlarmPackML. This was repeatedly done for all "events" at Machine, Equipment and Control module. The fb_AlarmPackML implemented the following attributes; "Event type", "Alarm trigger signal", "Jason attribute string", "Alarm type", "Reset signal for alarm" and "Alarm message" (see Section 2.12.1).

Machine module

M_AlarmInstances had four function block instances declared. One named fbAlarm_eStop, the second fbAlarm_Guard, the third fbAlarm_DeadGrip and lastly fbAlarm_Door.

fbAlarm_eStop monitored the alarms regarding emergency stop ("eStop"). Its alarm trigger signal was set to "bSafetyOk" equal false and "bSafetyFirstPowerCycle" equal true. In case "bSafetyOk" turned false, corresponding to the physical "eStop" button being pushed, the function block started larming. When "eStop" was reset, implying "bSafetyOk" equaled true, then the alarm got turned off. The "bSafetyFirstPowerCycle" was a boolean variable set to true during start up of the program. This would prevent the fbAlarm_eStop alarm to trigger when starting it.

fbAlarm_Guard tracks the alarms regarding the physical magnetic sensor. Its alarm trigger signal was set to "bGuardOk" equal false and "bSafetyFirstPowerCycle" equal true. When "bGuardOk" turned false, corresponding to the two magnetic sensors away from each other, the function block started larming. When together again the alarm would reset, implying "bGuardOk" to equal true which would turn the alarm off. The boolean "bSafetyFirstPowerCycle" was used in the same way as for fbAlarm_eStop function block.

fbAlarm_DeadGrip reacted to the alarms regarding the grips of the four industrial transportation robots. When the robots have grip, that results in a signal "bGrip" turning true. On the contrary when they do not, resulting in

"bGrip" turning false. The alarm trigger signal "bDeadGrip" turns to true in either of three different cases, which will be described in M_Safety.

fbAlarm_Door gave a response as an alarm connected to a simulation of an open door in Visual Components. The alarm trigger signal "bOpenDoor" was triggered to true or false through by the HMI menu. This will be described in detail in M_Safety.

In M_Safety the conditions from the function block alarms on Machine module level were written. Both fbAlarm_eStop and fbAlarm_Guard were checked with the same conditions and fbAlarm_DeadGrip and fbAlarm_Door separately.

For fbAlarm_eStop and fbAlarm_Guard a case statement was created which checked "eMachineStateSTS" that described the current machine state. Whenever the state was either "Undefined", "Aborting" or "Aborted", the EtherCAT safety variables were toggled true. These variables were "bRestart", "bRestart2" and "bErrorAck" presented in the EtherCAT safety section. When the current machine state was "Clearing", these variables were set to false. In any other occasion of the current machine state, the "bSafetyOk" and "bGuardOk" were checked. If one of them got a false value then a "MachinePMLCommand" was generated as "Abort" which would move the machine into "Aborting" state.

For fbAlarm_DeadGrip there were three safety conditions to be checked. Therefore, when the "bGrip" was true and either the "ESTOP" button was pressed (true) or the "bDoorOpen" true or the reached state was "Aborted". In anyone of these events the fbAlarm_DeadGrip would trigger. When that happened an "MachinePMLCommand" of Abort was sent and all modes were locked except "Manual" mode. Now the machine operator had to choose "Manual" mode when in "Aborted" state and move the product after choosing the jog axes to run. Once the machine operator has jogged the decided axes to new positions, pushing the "bGrip" button on the HMI unlocks all modes again.

For fbAlarm_Door there was only one safety condition considering "bOpenDoor". When "bOpenDoor" turned true the fbAlarm_Door Alarm started to trigger and the "Door angle" switched to 90 degrees in Visual Components indicating an "Unsafe" environment. When fbAlarm_Door trigger this would also send out "MachinePMLCommand" of Abort and lock all modes except "Manual" mode. Like the fbAlarm_DeadGrip case all axes have to be moved and a "bGrip" button pushed on the HMI.

Equipment module

M_AlarmInstances now had four 14 function block instances declared. First one named fbStartingSeqAlarm and the second one fbExecuteSeqAlarm. Among the remaining 12 they were declared as fbManualJogAlarmX, fbManualJogAlarmY and fbManualJogAlarmZ. Four each in X,Y and Z corresponding to the 12 in total virtual slave axes.

The fbStartingSeqAlarm was supposed to monitor alarms happening "Starting_Manual" state. The alarm trigger signal was set to "stStartingSeq.bFault".

If "stStartingSeq.bFault" was true, this would alarm fbStartingSeqAlarm. Correspondingly, when "stStartingSeq.bFault" was false, this would then turn off fbStartingSeqAlarm. This signal was written to only trigger if a coupling error appeared within the "Starting_Manual" state.

The fbExecuteSeqAlarm was supposed to track alarms happening in all modes for the "Execute" state. The alarm trigger signal was set to "stExecute.bFault". If "stExecuteSeq.bFault" was true, this would alarm fbExecuteSeqAlarm. Correspondingly, when "stExecuteSeq.bFault" was false, this would then turn off fbStartingSeqAlarm. This signal was written to only trigger if a PTP-movement, jogging or a halt error appeared within the "Execute" state.

For the fbManualJogAlarms in X,Y and Z, they were supposed to warn the machine operator whenever the transportation robots were close to each other or lower/upper boundaries in position.

The alarm trigger signals were set to "bAlarmTrigX", "bAlarmTrigY" and "bAlarmTrigZ" for each of the four slave axes in each axis direction. Whenever the signal turned true, the fbManualJogAlarm would start alarming and informing the machine operator what boundary exception had been reached. On the contrary, when the signal turned false, the fbManualJogAlarm would stop alarming.

Control module

Here, M_AlarmInstances had 17 function block instances declared in each Control module X,Y and Z. Four named fbAbortingSeqAlarm, four fbClearingSeqAlarm, one fbResettingSeqAlarm, four fbStoppingSeqAlarm and finally four aAxisAlarms.

The fbAbortingSeqAlarm was supposed to monitor alarms happening in all modes for the "Aborting" state. The alarm trigger signal was set to "stAbortingSeq.bFault". If "stExecuteSeq.bFault" equaled true, this would alarm fbAbortingSeqAlarm. Correspondingly, when "stExecuteSeq.bFault" equaled false, this would then turn off fbStartingSeqAlarm. This signal was written to only trigger if either a stop or a power-off error occurred within the "Aborting" state.

The fbClearingSeqAlarm was supposed to track alarms happening in all modes for the "Clearing" state. The alarm trigger signal was set to "stClearingSeq.bFault". If "stClearingSeq.bFault" equaled true, this would alarm fbClearingSeqAlarm. Correspondingly, when "stClearingSeq.bFault" equaled false, this would then turn off fbClearingSeqAlarm. This signal was written to only trigger if a reset, decoupling or a power-on error occured within the "Clearing" state.

The fbResettingSeqAlarm was supposed to react to alarms happening in all modes for the "Resetting" state. The alarm trigger signal was set to "stResettingSeq.bFault". If "stResettingSeq.bFault" equaled true, this would alarm fbExecuteSeqAlarm. Correspondingly, when "stResettingSeq.bFault" equaled false, this would then turn off fbStartingSeqAlarm. This signal was written

to only trigger if a PTP-movement or a coupling error appeared within the "Resetting_Production" or the "Resetting_Maintenance" state.

The fbStoppingSeqAlarm was supposed inform about alarms happening in the Manual mode for the "Stopping" state. The alarm trigger signal was set to "stStopingSeq.bFault". If "stStopingSeq.bFault" equaled true, this would alarm fbExecuteSeqAlarm. Correspondingly, when "stStopingSeq.bFault" equaled false, this would then turn off fbStartingSeqAlarm. This signal was written to only trigger if a halting or decoupling error appeared within the "Stopping_Manual" state.

The aAxisAlarms was supposed to yield alarms happening in all states and modes within the Control modules. The alarm trigger signal was set to "bAlarmTrig" and each one of the four slave axes in the control module got a separate trigger signal. If "bAlarmTrig" was true, this would alarm fbExecuteSeqAlarm. Correspondingly, when "bAlarmTrig" was false, this would then turn off fbStartingSeqAlarm. This signal was written to trigger only when an "NC"- error occurred in a state.

For M_Safety there was a need for conditions to handle when the aAxisAlarms function blocks should alarm. Whenever a "fbAxis.bError" would appear on an axis, this would activate the variable bAlarmTrig to true. Consequently this would then set aAxisAlarms into alarming mode. After the alarm had been reset, then "bAlarmTrig" was set to false which stopped aAxisAlarms from alarming.

3.11 HMI

The next step of the project was to create the HMI. This was done by using the software TwinCAT HMI (see Figure 30).

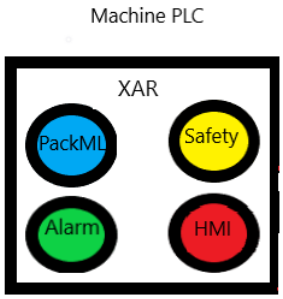

Figure 30. Inclusion of HMI in the system structure.

The first step in TwinCAT HMI was to create the desktop view. This was followed by creating several regions (in the desktop view) corresponding to the desired positions. For example, a region was created for the head menu-bar. Then the content was made for each region. In the case of the head menu-bar each button had different contents. The contents had to be connected to a region. The connection was done by choosing the attribute "Target Content".

The buttons were programmed so that upon pressing the desired region, it would display the corresponding content. This procedure was repeated for all the buttons.

It was also preferable to have an active/not active state of the buttons in the head menu-bar. For this to be done an integer variable was created and each button was given a (id)value ($0 \rightarrow 7$). Furthermore, the buttons were programmed so that if the "current number" was equal to the id of the button it would display the "active"-background (blue). Likewise it would display the "not active"-background (black) in the rest of the cases.

Another criteria was to be able to see the positions of the different axes. This was done via the ADS-communication. Firstly, a server had to be configured by assigning the correct AmsNetId of the CX5140. This made it possible to connect variables in the PLC to the HMI. The only thing needed to be done was making Textblocks and map the position of the axes (from the GVL).

3.11.1 Running the machine

When creating the buttons for the PackML-signals (Reset, Abort, Clear etc.) they were simply mapped to the PLC so that upon pressing it would set the corresponding signal to true.

One of the most important contents was the "sequence-content". Here the positions of the axes were displayed as well as the buttons for the commands (Reset, Abort, Clear etc) and the modes (Production, Manual and Maintenance). In this content a textblock was created to illustrate which current mode/state the machine was in. Furthermore, a region was created on which different contents, depending on which mode the machine was operating in, were to be displayed.

Manual mode

In the case of the Manual mode the sequence content was programmed to display only the Abort and Clear button since these were the only commands needed in this mode.

Furthermore buttons was created for the machine to jog forward/backward as well as buttons for choosing which axes to jog. The buttons on which would choose the axes was programmed that upon pressing, a green border would be displayed to indicate that the axis is gonna be able to be jogged. A button for turning the "Power on" (Jog-mode on) was also created. This button would make the program go into the execute-state so that the operator can jog the machine. Furthermore, a button that would simulate a constant contact controller was created. This button was to simulate whether the constant contact controller was pressed or not when jogging.

Additionally the function to choose the velocity in each axis (X, Y, Z) was created. This was done by making buttons for three different choices of velocity. The "High" velocity button would set the velocity in X to 300, Y to 100 and Z to 100. The "Medium" velocity button would set the velocity in X to 100 while

setting Y and Z to 50. Lastly the "Low" velocity button would set the velocity in X to 3 and Y and Z to 1.

Production mode

In Production mode the sequence content was to display the "Abort", "Clear", "Reset" and "Start" button, which were the desired buttons in the production mode. Moreover, one button to indicate whether a job is ready or not for the machine was created.

Four indicators (in shape of circles) were made with the purpose to display whether the machine was done with the "gripping" or not. This was done by mapping these indicators (circles) to the "bFeedbackGrip" values in the PLC. If the four transportation robots had completed the gripping the corresponding indicators would display a green background. On the contrary if the gripping was not completed the indicators would display a red background.

These indicators were later moved out of the "mode-region" to be displayed in every mode.

Maintenance mode

For the Maintenance mode the sequence content was exactly the same as for the Production mode.

In addition to this a "Step-forward"-button was implemented. This button would allow the operator to control the running of the machine. Basically, while the button was pressed, the machine would run the sequence (program) from the Production mode. When the button was not pressed the machine would stop the sequence.

Moreover, a "Homing"-button was created. Its purpose was to make it possible to jog the machine to the homing position at any time the machine was running.

To handle safety an error acknowledgement function was implemented in the sequence content. This was executed upon pressing the clear button. This button was to bring three flags of the "ESTOP"-block and Guard-block (Reset1, Reset2 and ErrorAck) from true followed by setting them to false. Furthermore, two rectangles were created to illustrate whether the safety was okay or not.

Lastly a button was created for the function to enter a visualization of which state the machine currently was in. This was done by choosing the target content upon pressing the button to "StateChart". The "StateChart" content was created by importing an overview picture of the PackML-states (see Figure 31).

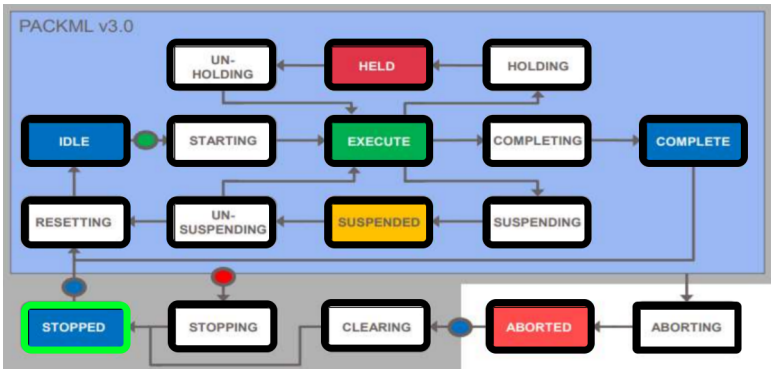

Figure 31. Illustration of current state of the machine.

3.11.2 Recipe management

To aid with the recipe management the example-program from Beckhoff was used. Firstly, a recipe type was made using the recipe management tab on the "TwinCAT HMI Configuration"-window. In this recipe type all desired parameters that were to be changed was added. These parameters could be things such as velocity, acceleration, deceleration, jerk, as well as positions for grabbing and leaving an object for the different transportation robots. Moreover, it contained homing-positions for the transportation robots.

For demonstration, two different recipes were created. One that was supposed to move smaller sheets of metal at a higher velocity as well as one for larger metal sheets at a lower velocity.

3.11.3 Internationalization

It was decided to make the HMI handle three languages: Swedish, English and Chinese. This was done by creating a localization file for each language. Thereafter, it was required to create keys for each word in the HMI that was desired to handle the different languages. For each key, the value was defined by the desired word in that specific language. An example of this would be a key "Prod" that have the value "Production" in English and "Produktion" in Swedish.

When the languages were implemented the only thing left to be done was to make the altering of the languages an option in the HMI. To do this three buttons (with the flags of the three languages as background) were created. Upon clicking on one of these buttons it would set the parameter "SetLocal" to the corresponding localization file of the language.

3.11.4 Publishing the HMI

The last step when doing the HMI was to publish it to the HMI-server. In this case the HMI-server was the PLC device (CX5140). For the PLC to be able to be the server software (TF2000-HMI-Server) from Beckhoff had to be

downloaded. When the installation was finished the CX5140 now could host a HMI-server.

Furthermore, the publishing was done in the TwinCAT HMI program. This was executed by simply choosing which AMSNETid the HMI-server was located on and then publish the solution. Once the solution was published, an URL was achieved which was the link to the web-application of the HMI. It was now possible for anyone on the same net to use the web-application (via ADS-connection).

3.12 Simulation of 12-axis machine

Eventually, Visual Components assisted through a python script with code to change the coordinates of a pallet. Now the pallet could interact with the 12-axis machine in terms of being "lifted up" and "dropped off". In addition, there were two conveyor belts and a pallet feeder added to the total simulation. For the 12-axis machine to interact with the pallet, the Machine operator had to step forward through the PackML states until "Execute_Production" was reached.

At first, the pallet arrived at an "initial position" on the first conveyor belt. It was supposed to represent an input pallet from an pallet feeder machine connected to the conveyor belt (see Figure 32).

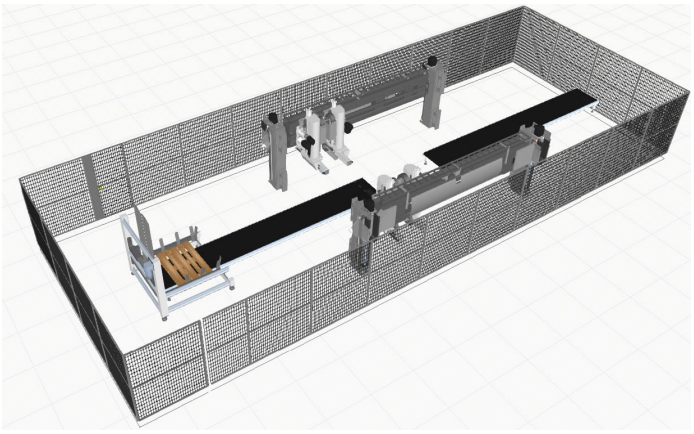

Figure 32. 12-axis machine at "waiting position and pallet on its "Initial position".

The pallet got transported along the first conveyor belt until it reached a "Grabbing position" on the X-axis. On this position a sensor signal connected to a PLC variable, "bSensor", got set to true and informed the 12-axis machine that a "Job was ready". From this the 12-axis machine could go down and pick up the pallet from the first conveyor belt (see Figure 33).

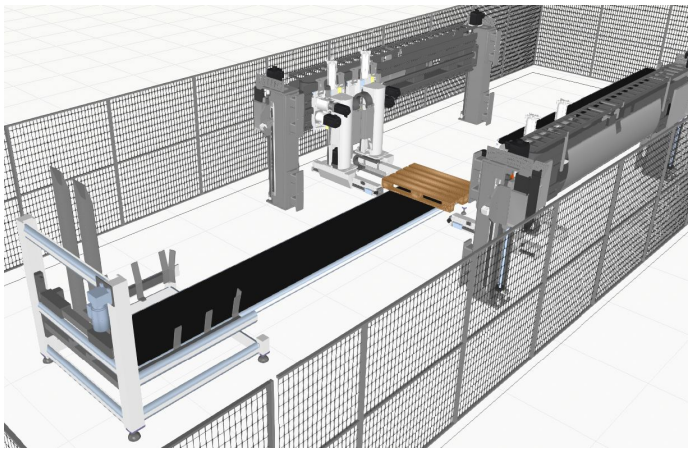

Figure 33. 12-axis machine and pallet at "grabbing position".

The 12-axis machine would then move to a "destination position" and drop of the pallet at the second conveyor belt. Since the pallet now had left the sensor, the connected variable "bSensor" would turn false. This informed the 12-axis machine that a "Job was not ready" (see Figure 34).

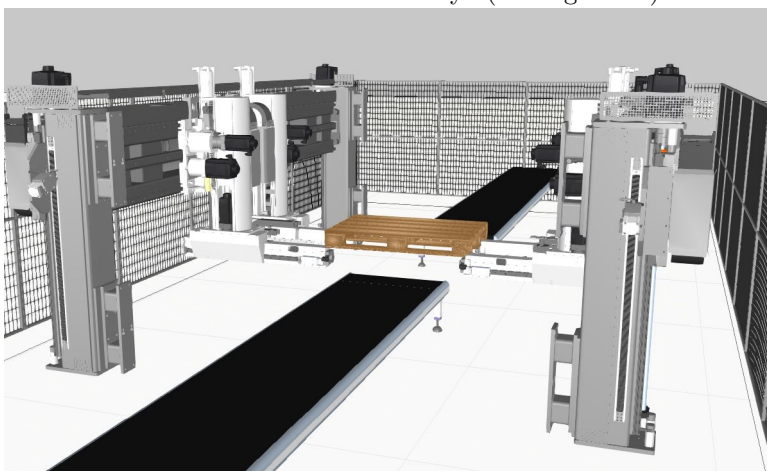

Figure 34. 12-axis machine and pallet moving towards "Destination position".

At the "Destination position" the 12-axis machine dropped of the pallet. The 12-axis machine then returned to the "waiting position". This while the pallet started moving towards the "travel position" at the second conveyor belt (see Figure 35).

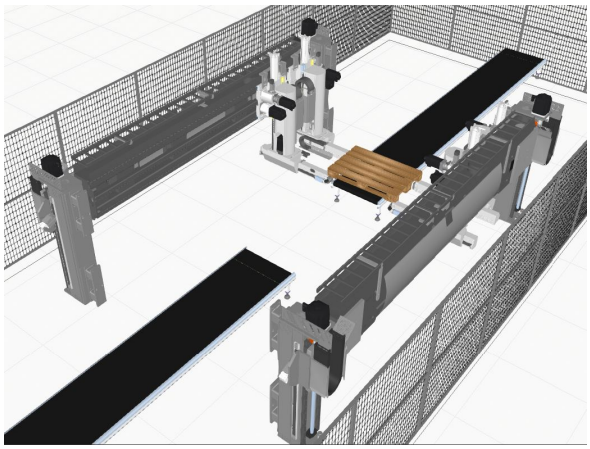

Figure 35. 12-axis machine and pallet at "destination position".

When the pallet eventually reached the "traveling position" the simulation would repeat itself. Meaning the pallet would start off at its "initial position" and 12-axis machine at "waiting position" (see Figure 36).

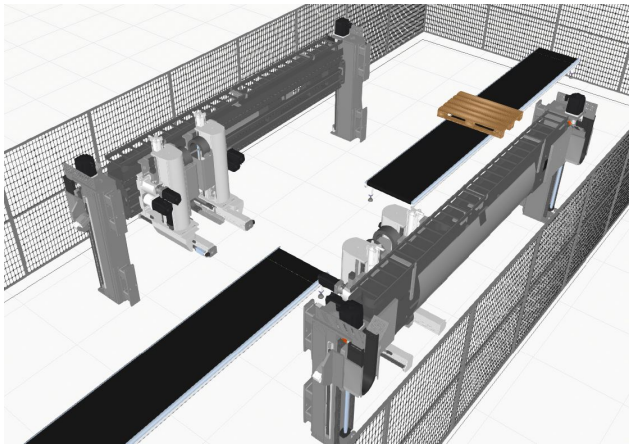

Figure 36. 12-axis machine at "waiting position" and pallet moving towards "Traveling position".

4 Result

The result of this Master Thesis was a successful implementation of Virtual Commissioning of a machine consisting of four industrial transportation robots. The total 12 virtual slave axes of the machine were coupled to three virtual master axes in the PLC. The functionality of the machine was simulated in Visual Component. The structure of the code is written in TwinCAT according to the PackML-structure. Furthermore, a EtherCAT-simulation is used for realising the signals for the 12-axis machine, which represent the I/O:s of the system.

The program structure was implemented with alarm handling as well as safety. Within the safety an emergency stop and a guard were implemented. Moreover, a constant contact controller also existed.

Additionally, a HMI with functions such as recipe management, displaying axis positions, changing mode, running the machine, simulating errors etc. now exists.

The hardware for the project consisted of a Laptop, a CU2508, a CX5140 PLC, two EL6910 Safety Cards and an Emergency Stop (see Figure 37).

On the Laptop all the code was written (XAE) in TwinCAT and also the simulation program Visual Components was used to visualize the 12-axis machine.

The CU2508 EtherCAT switch connected the code from the Machine PLC-project and the Simulation PLC-project through an EtherCAT cable.

For the CX5140 PLC both the Machine PLC-project and the Simulation PLC-project were running and compiled (XAR).

The EL6910 Safety cards were attached to the CX5140 PLC and interconnected by input and output pins.

The Emergency stop button was connected to one of the two EL6910 Safety cards. The button responded by both stopping and turning of the 12-axis machine when pushed down.

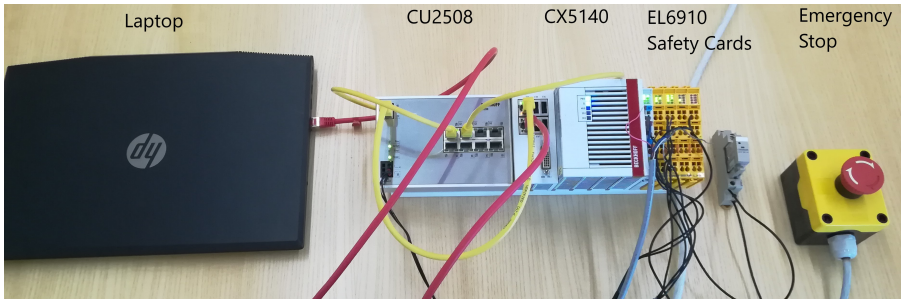

Figure 37. Electronic hardware for the 12-axis machine

5 Discussion

5.1 ADS-communication

When connecting the PLC to VC via ADS-communication some issues occurred. One of the issues was that VC could not find the PLC for some reason. The support at VC was then contacted trying to solve the problem. With help from Beckhoff the complication was that the CRL (Communication Reference List) searched for the wrong version of TwinCAT (4.1.16.0) when the actual version was 4.2.169.0. When correcting this (by altering the configuration file of VC) an ADS-connection could be established.

5.2 The model of the machine in Visual Components

The separation of the axes turned out to be a bit more complicated than expected. Since a step-file was used, which basically is a blueprint, all the different components had to be chosen in great detail. This was problematic because firstly it was hard to distinguish which components that had to be linked. Secondly, since the modelling tool was very detailed, it was easy to miss internal (or small) components, such as screws. This made the search for every component time-consuming and had to be redone several times.

Another issue that appeared in terms of the 3D-model was to place the orientation of the transportation robots origin. The idea was to have a positive orientation for all axis directions, so the coordinates of the robots were moving between zero and positive values. This would hopefully make it easier for the machine operator to adjust offsets between the robots and make less mistakes. Initially, it was decided to place the transportation robots furthest to the left on the x-axis. That would set the origin 0 and also the lower boundary. Since the length of the machine was 4 m, this was set to be the upper boundary. It was realised that the closest distance between the adjacent transportation robots were 0.7 m. Therefore the upper limit for the second and fourth transportation robot became 4 m. Correspondingly, the upper limit for the first and the third transportation robot was set to 3.3 m (see Figure 38).

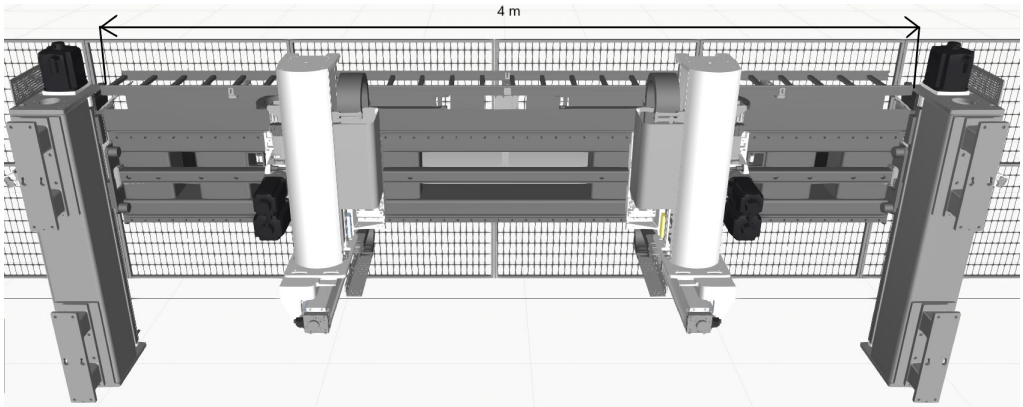

Figure 38. The 3D-model for one side of the 12-axis machine.

For further visualization it was tested to attach a pallet to the 12-axis machine while in movement. This to simulate a realistic production-line where a sensor conveyor sent a pallet for the transportation machine to pick up. This was tested using the "Snap" button in the Modelling toolbar inside VC. Although, this did not work since it required further knowledge of the program. The support of VC was contacted, but they required the simulation file to the transportation machine (see Figure 39).

However, the support of Visual Components was eventually contacted once more and they gave a python script for assigning 3-D coordinates to an object. This was proven useful since now it was possible to attach a pallet to the 12-axis machine. A virtual PLC-axis in X-direction was connected to the pallet so it could move in a realistic manner. Furthermore, the pallets 3D-positions was in some occasions copying the positions of the 12-axis machine when they were connected.

It was noted that only one pallet could be uniquely connected to the corresponding 3D-coordinates from the PLC project. Consequently, since the lack of knowledge how to use more pallets, only one pallet was used for the simulation.

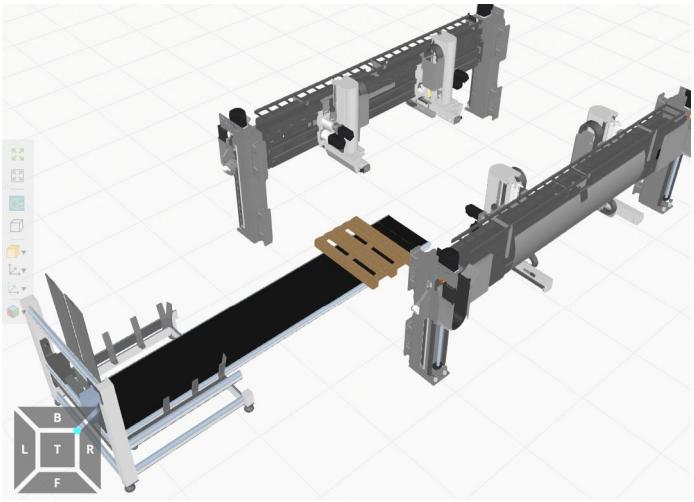

Figure 39. Visualization of the 12-axis machine with a conveyor belt.

5.3 Coupling of axes in TwinCAT

It was later on discovered that the MC2 function-block MC_GearOut block had to be implemented so coupled axes could gear out. Therefore, this function-block also had to be implemented in each control module like for MC_GearIn. An issue that was encountered was that the "InGear" signal had to be set down from the MC_GearIn block before the "OutGear" signal could be triggered. Therefore both MC_GearIn and MC_OutGear had to be called upon when gearing out axes.

5.4 PackML-structure

In regards to the implementation of PackML for this project it turned out to be a difficult task. Since the theory of the subject was quite substantial it was a lot to process. Even though aid was received through meetings and previous PackML-projects, it was still quite a big obstacle. The example-project that was received was very detailed which resulted in hard interpretation of the different sections of the structure. Furthermore, it was troublesome finding other easier example project with PackML for guidance.

However, after some weeks of analyzing the structure as well as executing tests on the program it started to make a lot more sense. Initially, the tests were simply writing some code in some of the states and observe how the code was executed. This could be for instance writing global variables that worked as integer counters. Whenever inside a PackML state the corresponding counter for this state would count up. Through this procedure it was easy to find if a state was reached and if it was active or not. When active the state counter would increment and when it was not it stopped counting.

It was eventually discovered that the PackML three level hierarchy compiled the code in a top-down manner. Firstly, the machine states were compiled and then the equipment states and finally the control states. However, if a machine state wrote a "MachinePMLCommand" to reach another state both the equipment- and the control states would follow it to the next state. Although, when an equipment state wrote a "EquipmentPMLCommand" only the corresponding control state would follow it.

5.5 Structs

It was proven difficult to use the structs data types within TwinCAT. A possible reason could be the weak background of object-oriented programming within the PLC. The structs were not connected to any global variables which could be reached everywhere within the PLC project. Instead, it was required to create instances of function blocks within a folder or a file and then call upon a method that implemented a struct. Thereafter, it was needed to call the method upon the function block or file and set values within it to make it work. To use the GearIn-method for a struct was complex to implement since it could not be reached once called upon from an instance declaration of ST_MC_InGear. Instead, a new instance of the function block which contained the method from another folder had to be declared, fbAxis_PTP.

5.6 HMI - Internationalization

Even though the internationalization was fairly simple once understood it would have been even more simple if this was handled right from the beginning of the creation of the HMI. Sadly this was not the case here. The function to handle several languages came up during the later part of the project. At that time of the project the HMI was nearly finished (with already defined names of buttons etc.). The names (where internationalization was desired) of the buttons/textblocks etc. had to be replaced with the corresponding key. However, this was fixed without any further complications.

5.7 EtherCAT Safety

It was proven important to take down all the flags in the "Clearing" states so the "ESTOP" would not be in-activated when the "Resetting" state was reached. If this was not done it was impossible to gear in all the axes and moving the axes to the homing positions. This would maybe not have been necessary if the MC2_Homing function block would have been implemented. Mainly because it does not require any MC2_MoveAbsolute calls on the axes.

5.8 Alarm Handling

For the alarm handling it was troublesome finding the proper data structures to start off. It was eventually required to create a new function block FB_AlarmPackML. The FB_AlarmPackML implemented both TC_Event classes from the Event Logger and the St_Alarm from PackML.

6 Conclusions

For simulating the function of the robots, VC made it easy for the programmer to observe the movement boundaries in 3D-space. Therefore it went fast to calibrate new values for variables declared in TwinCAT to gain desired machine behaviour.

On each axis there were four slave axes per each X,Y and Z master axis. Since all code was driven through the three virtual master axes, the coding and safety management became a lot easier than if the axis had been coded separately.

To implement the PackML-structure required a lot of theoretical understanding. After the theoretical knowledge had been gained, the programming became more organized and easy to follow.

To declare a new function block for the alarm handling was proven challenging. Once this was done, new function block instances could be created simply within the Machine, Equipment and Control Module levels.

The safety "ESTOP" worked as intended with a "physical" press button that had to be manually reset. In terms of the "Guard" it was simulated through an open or closed door in VC. This would imply that a realistic situation can be created through a door with magnetic sensors.

For the EtherCAT simulation it was efficient to use an extra PLC program simulating the I/O:s from the Simulation PLC. By this, physical grips for the transport robots could be simulated through VC into the Machine PLC.

Furthermore, the HMI with the buttons and coordinates gave the machine operator a simple display to control the machine from. The recipe functionality in the HMI had to be manually activated in the "Aborted" state. This was independent of the machines' current mode (Production, Manual or Maintenance). However, the machine operator had to remember pressing the Abort command before changing recipe.

7 Further Work

For further work there are some things to consider. The possibilities with the EtherCAT-simulation is somewhat endless.

To give an example it would be of much interest to simulate the actual behaviour of the drives. This would be possible by adding a simulated drive to the PLC-project and take that into account when writing the code.

Another example would be to have a model of the actual signal from the real system. The models could be for things such as sensor signals. These models could be developed in Simulink which could be added to the Simulation PLC.

There are a lot of these simulations that can be created (in principle for the whole machine). The optimal case would be a Simulation PLC which replicates the exact behaviour of the real system.

8 References

[1]. Searcherp (2018). *Virtual Commissioning.* https://searcherp.techtarget.com/definition/virtual-commissioning. (collected, 2019-02-01)

[2]. Assemblymag (2017). *How to Virtually Commission an automated manufacturing system.* https://www.assemblymag.com/articles/938333-how-to-virtually-comission-an-automated-manufacturing-system. (collected, 2019-02-01)

[3]. AP&T_Company (2019). *ABOUT AP&T* https://www.aptgroup.com/company/about-apt (collected, 2019-02-06)

[4]. Beckhoff_Company (2018). *Beckhoff Automation.* https://www.beckhoff.com/english/beckhoff/default.htm. (collected, 2019-02-04)

[5]. AP&T_Automation (2019). *Automation* https://www.aptgroup.com/solutions/product-range/automation (collected, 2019-02-06)

[6]. Beckhoff_ADS (2019). *ADS Communication.* https://infosys.beckhoff.com/english.php?content=../content/1033/bc9000/html/bt_ethernet%20ads%20potocols.htm&id=. (collected, 2019-02-04)

[7]. VisualComponents_ConnectPLC (2019). *Connect a local TwinCAT PLC* http://academy.visualcomponents.com/lessons/connect-a-local-twincat-plc/. (collected, 2019-02-04)

[8]. VisualComponents_About (2019). *About us* https://www.visualcomponents.com/about-us/. (collected, 2019-02-04)

[9]. Siemens (2018). *Virtual Commissioning ger kortare time-to-market.* https://w3.siemens.se/home/se/sv/industry/nyheter/pages/virtual-commissioning-ger-kortare-time-to-market.aspx#content. (collected, 2019-02-01)

[10]. VisualComponents_VC (2016). *What is Virtual Commissioning* https://www.visualcomponents.com/insights/miscs/increasing-control-software-quality-with-virtual-commissioning/. (collected, 2019-02-01)

[11]. CTH (2019). *Simulation-based verification of PLC programs.* http://publications.lib.chalmers.se/records/fulltext/195493/195493.pdf. (collected, 2019-02-05)

[12]. Beckhoff_WindowsControl (2019). *The windows control and automation technology.* https://beckhoff.com/english.asp?twincat/default.htm. (collected, 2019-02-05)

[13]. CollinsDictionary (2019). *Defintion of 'production line'.* https://www.collinsdictionary.com/dictionary/english/production-line. (collected, 2019-05-30)

[14]. Beckhoff_E-CATSimulation (2019). *Tc3 Ethercat simulation.* https://www.beckhoff.com/english.asp?twincat/te1111.htm. (collected, 2019-05-30)

[15]. VisualStudio_IDE (2019). *Microsoft Visual Studio.* https://en.wikipedia.org/wiki/Microsoft_Visual_Studio. (collected, 2019-04-14)

[16]. VisualStudio_IsolatedShell (2019). *Microsoft Visual Studio isolated shell.* https://visualstudio.microsoft.com/vs/older-downloads/isolated-shell/. (collected, 2019-04-14)

[17]. PLCdev (2009). *Defintion of a PLC.* http://www.plcdev.com/definition_of_a_plc. (collected, 2019-03-25)

[18]. Beckhoff_CX5140. *CX 5140 | Embedded PC with Intel ATOM processor.* IPC, Motion and Automation, first ed. Beckhoff New Automation Technology, page 230, 2018

[19]. Beckhoff_E-CATG (2019). *Ethercat.* https://download.beckhoff.com/download/document/catalog/Beckhoff_EtherCAT_G_e.pdf. (collected, 2019-05-07)

[20]. Beckhoff_PLC (2014). *PLC and motion control on the PC.* https://www.beckhoff.com/english.asp?twincat/einlei1.htm/. (collected, 2019-03-25)

[21]. ElectronicDesign (2016). *Whats the difference between jitter and noise?* https://www.electronicdesign.com/test-measurement/what-s-difference-between-jitter-and-noise. (collected, 2019-05-31)

[22]. Beckhoff_FunctionBlocks (2019). *Object Function blocks.* https://infosys.beckhoff.com/english.php?content=../content/1033/tc3_plc_intro/9007201785020555-1.html&id=. (collected, 2019-05-07)

[23]. Beckhoff_Method (2019). *Object Method.* https://infosys.beckhoff.com/english.php?content=../content/1033/tc3_plc_intro/9007201785048459-2.html&id=. (collected, 2019-05-06)

[24]. Beckhoff_Action (2019). *Object Action.* https://infosys.beckhoff.com/english.php?content=../content/1033/tc3_plc_intro/2530340747.html&id=. (collected, 2019-05-06)

[25]. Beckhoff_GVL (2019). *Global Variables.* https://infosys.beckhoff.com/english.php?content=../content/1033/tcplccontrol/html/TcPlcCtrl_ResGlobVar.htm&i=. (collected, 2019-05-06)

[26]. Beckhoff_Struct (2019). *Structures (struct).* https://infosys.beckhoff.com/english.php?content=../content/1033/tcplccontrol/html/tcplcctrl_struct.htm&id=. (collected, 2019-03-29)

[27]. MIT (2019). *Reading 14: Interfaces.* http://web.mit.edu/6.005/www/fa15/classes/14-interfaces/. (collected, 2019-03-29)

[28]. Beckhoff_NCAxes (2019). *TwinCAT NC axes.* https://infosys.beckhoff.com/english.php?content=../content/1033/tcncgeneral/html/tcncaxis.htm&id=2765368370592350287. (collected, 2019-05-06)

[29]. Beckhoff_Motion (2018). *Motion function blocks.* https://www.beckhoff.com/english.asp?twincat/twincat-3-functions.htm?id=1905053018901109. (collected, 2019-03-22)

[30]. EtherCAT_Company (2019). *Ethercat technology group.* https://www.ethercat.org/en/tech_group.html. (collected, 2019-04-18)

[31]. Beckhoff_E-CATFast (2018). *Beckhoff Ethercat components: Fast.* https://www.beckhoff.com/ethercat/. (collected, 2019-02-05)

[32]. EtherCAT_Function (2019). *Ethercat.* https://www.ethercat.org/en/technology.html. (collected, 2019-04-18)

[33]. Beckhoff_SafetyCard (2019). *EtherCAT safety-card* https://www.beckhoff.com/EL1904/. (collected,2019-05-06)

[34]. OMAC (2019). *PackML unit/machine implementation guide.* PackML_Unit_Machine_Implementation_Guide-V1-00.pdf. (collected, 2019-02-06)

[35]. Beckhoff_Framework (2019). *TwinCAT framework ideas.* TwinCATCodeLayoutV2.docx. (collected, 2019-02-06)

[36]. Yaskawa (2012). *PackML* PackML_Modes_Yaskawa.pdf. (collected, 2019-05-07)

[37]. ControlEngineering_Alarm (2018). *Alarm management: 6 hazards, 4 strategies.* https://www.controleng.com/miscs/alarm-management-6-hazards-4-strategies/. (collected, 2019-05-31)

[38]. Beckhoff_EventLogger (2019). *Event Logger.* https://download.beckhoff.com/download/Document/automation/twincat3/TC3_EventLogger_EN.pdf. (collected, 2019-05-08)

[39]. SquareSpace (2019). *JSON.* https://developers.squarespace.com/what-is-json. (collected, 2019-05-08)

[40]. Beckhoff_Alarm (2019). *Alarm.* https://download.beckhoff.com/download/Document/automation/twincat3/TwinCAT_3_PLC_Lib_Tc3_PackML_V2_EN.pdf. (collected, 2019-05-08)

[41]. ControlEngineering_Drives (2018). *Drives software programming – using PLC or drive custom programming?.* https://www.controleng.com/

miscs/drives-software-programming-using-plc-or-drive-custom-programming/. (collected, 2019-02-08)

[42]. Cenito (2012). *What is HMI?* http://www.cenito.se/sv/hmi/. (collected, 2019-02-05)

[43]. InductiveAutomation (2012). *What is HMI?* https://inductiveautomation.com/resources/misc/what-is-hmi. (collected, 2019-04-11)

[44]. Beckhoff_HMI (2019). *TwinCAT HMI.* https://download.beckhoff.com/download/document/automation/twincat3/TE2000_TC3_HMI_EN.pdf. (collected, 2019-04-11)

[45]. Automationdirect (2019). *Automationdirect.* https://library.autoamtion.direct.com/learn-about-hmi-recipes/. (collected, 2019-04-18)