UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

FINAL THESIS (TFE)

Bachelor's degree in Industrial Electronics and Automatic Control Engineering

DEVELOPMENT OF A FUNCTION BLOCK LIBRARY TO COMMAND OMRON COLLABORATIVE ROBOT FROM AN EXTERNAL CPU

Thesis and Annexes

Author: SÁNCHEZ BOLI, RUBÉN
Director: GÁMIZ CARO, JAVIER FRANCISCO

Summons: June 2021

Abstract

Collaborative robots are designed to be easily programmed by non-expert operators, but when the application requires the robot to be integrated and communicate with the rest of the machine, being the robot commanded by external controllers, the programming becomes more complex hence certain expertise and knowledge in robot programming is required. The target of this project is providing to non-skilled operators, a library with a set of Function Blocks to command the Omron collaborative robot from an external Omron controller without needs to learn how to build complex TCP/IP (*Transmission Control Protocol / Internet Protocol*) communication frames. The library is designed following the international standard IEC-61131 for Programmable Controllers and compliant PLCopen guidelines (worldwide association for industrial programming control).

In this thesis, it is described the robot software program, the protocol to send commands from external devices to the robot, the algorithm inside the Function Blocks, an example of Pick and Place sequence program, and a graphical interface for touch-screen to use the library and control the robot defining the parameters through drop-down lists and data entry fields, facilitating the user to use the Function Blocks. After the pertinent tests with real hardware and once the tests for debugging and implementation of functionality improvements have been completed, a package of Function Blocks has been obtained whose input parameters have a predefined range that is verified internally before sending the command to ensure this is correct and will be executed by the robot.

Resum

Els robots col·laboratius estan dissenyats per a ser programats fàcilment per operadors no experts, però quan l'aplicació requereix que el robot s'integri i comuniqui amb la resta de la màquina, en ser el robot comandat per controladors externs, la programació es torna més complexa, per la qual cosa es requereix una certa experiència i coneixement en programació de robots. L'objectiu d'aquest projecte és proporcionar als operadors no qualificats una llibreria amb un conjunt de Blocs de Funció per a controlar el robot col·laboratiu Omron des d'un controlador extern Omron, sense necessitat d'aprendre a construir complexes trames de comunicacions TCP/IP (*Transmission Control Protocol / Internet Protocol*). La llibreria està dissenyada seguint l'estàndard internacional IEC-61131 per a Controladors Programables i les directrius de PLCopen (associació mundial per al control de programació industrial).

En aquesta tesi, es descriu la programació del programari del robot, el protocol per a enviar comandos des de dispositius externs al robot, l'algoritme dels Blocs de Funció, un exemple de programa de seqüència *Pick and Place*, i una interfície gràfica per a pantalla tàctil per a usar la llibreria i controlar el robot definint els paràmetres a través de llistes desplegables i camps d'entrada de dades, facilitant així a l'usuari l'ús dels Blocs de Funció. Una vegada realitzades les proves pertinents amb maquinari real i finalitzades les proves de depuració i implementació de millores de funcionalitat, s'ha obtingut un paquet de Blocs de Funció on els paràmetres d'entrada tenen un rang predefinit que es verifica internament abans d'enviar el comando per a assegurar que aquest és correcte i serà executat pel robot.

Resumen

Los robots colaborativos están diseñados para ser programados fácilmente por operadores no expertos, pero cuando la aplicación requiere que el robot se integre y comunique con el resto de la máquina, al ser el robot comandado por controladores externos, la programación se vuelve más compleja, por lo que se requiere cierta experiencia y conocimiento en programación de robots. El objetivo de este proyecto es proporcionar a los operadores no calificados una librería con un conjunto de Bloques de Función para controlar el robot colaborativo Omron desde un controlador externo Omron, sin necesidad de aprender a construir complejas tramas de comunicaciones TCP/IP (*Transmission Control Protocol / Internet Protocol*). La librería está diseñada siguiendo el estándar internacional IEC-61131 para Controladores Programables y las directrices de PLCopen (asociación mundial para el control de programación industrial).

En esta tesis, se describe la programación del *software* del robot, el protocolo para enviar comandos desde dispositivos externos al robot, el algoritmo de los Bloques de Función, un ejemplo de programa de secuencia *Pick and Place*, y una interfaz gráfica para pantalla táctil para usar la librería y controlar el robot definiendo el parámetros a través de listas desplegables y campos de entrada de datos, facilitando así al usuario el uso de los Bloques de Función. Una vez realizadas las pruebas pertinentes con *hardware* real y finalizadas las pruebas de depuración e implementación de mejoras de funcionalidad, se ha obtenido un paquete de Bloques de Función cuyos parámetros de entrada tienen un rango predefinido que se verifica internamente antes de enviar el comando para asegurar que éste es correcto y será ejecutado por el robot.

Acknowledgements

To Irene, who has been by my side through good and bad times throughout my entire bachelor's degree.

Glossary

Hereunder it is described in alphabetical order the list of acronym names used along this thesis. Even though all these contractions are described the first time they appear in this document, this section can be considered as brief dictionary.

Cobot Collaborative Robot

FB Function Block

FIFO First In, First Out

GUI Graphical User Interface

HMI Human Machine Interface

MAC Machine automation Controllers

OEM Original Equipment Manufacturer

PLC Programmable Logic controller

PnP Pick and Place

PTP Point to Point

SI System Integrators

SoC Separation-Of-Concerns

TCP Tool Center Point (robot tip)

TCP/IP Transmission Control Protocol / Internet Protocol

VAC Voltage Altern Current

VDC Voltage Direct Current

GND Ground

Index

1. Prologue

1.1. Origen of the thesis

Most of cobot (Collaborative Robot) manufacturers design their product simple as much as possible with the target of becoming accessible for non-expert users. Since the performance and complexity of collaborative robots are lower than industrial robots, high skilled people are not required to make them work. Cobot is conceived as stand-alone unit providing flexibility in those applications with repetitive tasks in high-mix and very low-volume demanding regular production change-over. Likely this is the tendency, but not all applications require regular production change-over.

There are applications in which most of the production is done in an industrial machine controlled by only one PLC (Programmable Logic Controller) equipped with fences and other safety devices. In those machines, some areas require fast production but human interaction in other zones, therefore cobots are used to allow the operator to entry to the robotic cell without stopping the production. Both, cobot and human, are working in the same workspace with no risks for the human because cobot supports power and force limitation for collaborative operations. But the interesting point is that all components in the machine are controlled by the same PLC, thus instead of just exchanging variables or signals between PLC and Cobot, the proper way to control the robot is sending movement commands, something not trivial with Omron Collaborative Robots because nowadays it is not provided an integrated solution and the user has to create their own programming to send the commands properly from and external PLC.

1.2. Motivation

Currently, there is a high demand in the industry to develop applications, like palletizing or machine tending, with Omron collaborative robots requiring centralized PLC control. Therefore, the aim of this project is providing to customers, like OEM (Original Equipment Manufacturer) or SI (System Integrators), a library with a set of Function Blocs to command the robot without needs to learn how to build complex communications frames. Some of the benefits of this library are:

- Easy to program.
- Fast implementation.
- Reliability.
- Standard solution.

Another emerging kind of application is a collaborative robot mounted on top of a mobile robot, also known as Mobile Manipulator, this concept arises from the need to move the robot to different work areas instead of having a robotic cell and transporting the workpieces to the robotic cell. This avoids having many transportation lines, like belts or conveyors, for moving the goods inside the factory.

In such robotics system design, there is a PLC working as master of both robots, the slaves. With the development of this library, the cobot can be easily controlled. Obviously, another different library would be required to control the mobile robot.

Figure 1.2.1 **-** Omron Mobile Manipulator (source: own).

2. Introduction

2.1. Thesis objectives

Following the guidelines proposed by PLCopen [1], it is intended to create a library containing a set of FBs (Function Block) encapsulating, in the correct syntax protocol, the string frame package that Omron cobot requires to by commanded from an external device by receiving TCP/IP messages.

The main objective is providing a set tools to easily command the robot from the PLC. Therefore, the user has no needs to be concerned about understanding the protocol required by the robot nor investing time developing the programming and debugging.

2.2. Thesis scope

Even though the objective is controlling a robot from an external device, the project scope is PLC based, providing to the user the following products:

- FB Library to control Omron cobots from external Omron PLC containing following functions:
    - Move the robot to an absolute position, in cartesian or joint coordinates.
    - Move the robot to a relative position, in cartesian or joint coordinates.
    - Move the robot with a circular trajectory in cartesian coordinates.
    - Change robot base coordinates.
    - Change robot tool offset, weight, and inertia.
    - Pause and resume robot program.
- Description for all FB's, inputs/outputs datatypes and guidelines for their usage.
- Sysmac Studio project with one example of each FB and a PnP (Pick and Place) application sample with absolute position movements.
- HMI (Human Machine Interface) application for an intuitive usage of the robot with all the FB created and a PnP application example.

3. System Setup

3.1. Hardware

The setup arranged to design, develop, debug, and test the correct functionality of the FB library is composed by:

- Omron TM-series [2] collaborative robot is a 6-axis model with power and limiting function, featuring simple programming and integrated camera close to the TCP (Tool Center Point). Cobot is provided with its own controller in which power supply, digital and analogue inputs and outputs, safety functions and amplifiers are included. This robot is the target device which interprets the messages and executes the instructions when received from Omron NJ-series PLC.

Figure 3.1.1 **-** Omron TM-Series collaborative robot and controller (source: [10]).

- Omron NJ-series [2] PLC is a machine controller for logic sequence, safety, motion, and database connection, among others, with 500μs of scan cycle time. The FB library has been developed to be used for this controller.

Figure 3.1.2 **-** Omron NJ-Series PLC (source: [8]).

- Omron NA-series [2] HMI is the innovative Omron touchscreen enabling faster control and monitoring. It provides to the user easy and intuitive 9-inches interface to command the robot using the FB's library.

Figure 3.1.3 - Omron NA-Series HMI (source: [13]).

3.2. Software

3.2.1. TMflow

TMflow is the graphical environment that provides to users a complete interface for Omron TM-series collaborative robot motion and logic programming environments. TMflow uses a graphical flow chart language to process logic and robot motion.

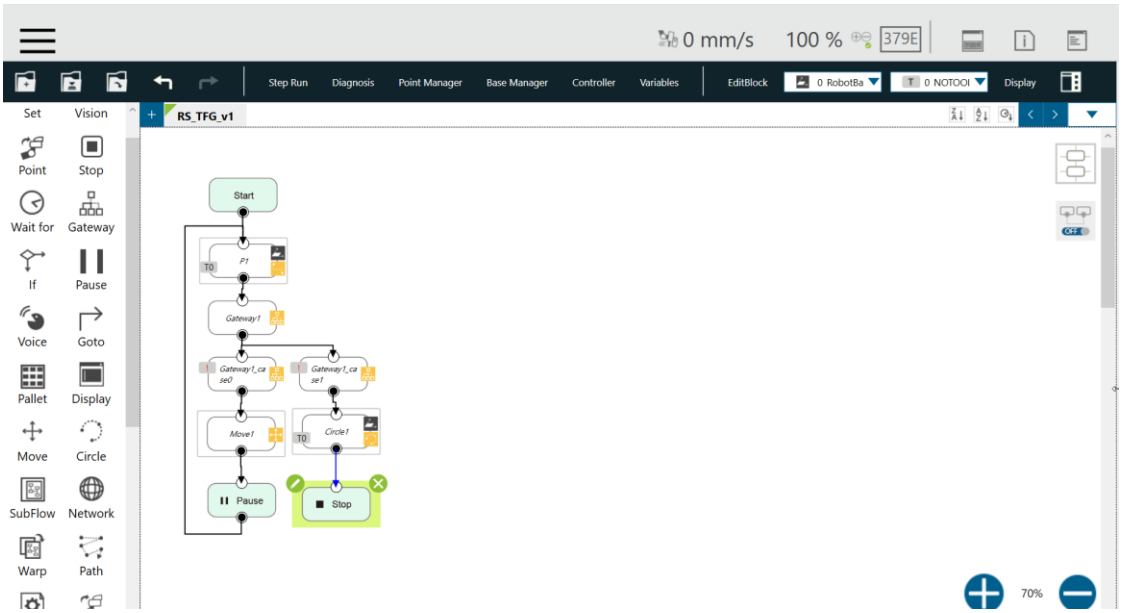

Figure 3.2.1 - TMflow software for robot programming (source: own).

On the right of the project editor (figure 3.2.1) there is a toolbox with all the functions available. Those functions are encapsulated in boxes called *nodes*, by drag and drop, the operator can build the program. The programming is based on flowchart composed by nodes providing certain functionality like: Motion nodes, logic nodes, conditional nodes, and communication nodes. The flow is determined by the arrows connecting the nodes that are executed sequentially. One of the most relevant nodes allowing communications from external device is the one called *Listen Node*.

Safety parameters for power and limiting function can be adjusted: maximum force, speed, position of each joint, etc. Communication settings are also configurable.

3.2.2. Sysmac Studio

The Sysmac Studio provides an integrated development environment to set up, program, debug, and maintain Omron PLC units for motion, logic, safety, drives, vision, robots, and HMI. Sysmac Studio is fully compliant with open standard IEC 61131-3 [4] and supports Ladder, Structured Text, and Function Block programming.

Sysmac Studio allows to program the HMI device with VB.NET (Visual Basic .NET). It is an object-oriented programming language implemented in .NET Framework and developed by Microsoft.

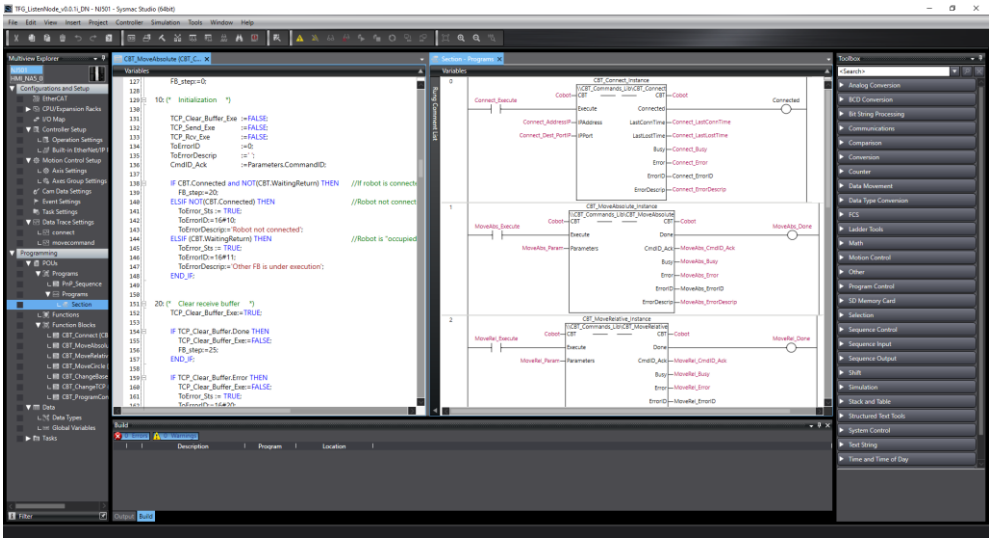

Figure 3.2.2 – Sysmac Studio software for PLC programming (source: own).

3.3. Communications Architecture

The connection between robot, PLC, HMI, and personal computer is through EtherNet/IP fieldbus that encapsulates TCP/IP protocol frames as user data.

Each device is equipped with, at least, one ethernet port which is connected to the Omron Industrial Ethernet Switching Hub to provide communication between any device in the network. To be able to configure the Ethernet port settings, it is needed to connect the laptop with each device individually. HMI and PLC ports have been configured with Sysmac Studio software; Cobot port has been configured with TMflow software. In any case, the software was previously installed in the laptop.

Ethernet network cables are Category 6 (Cat 6), standardized twisted pair cable achieving 250MHz.

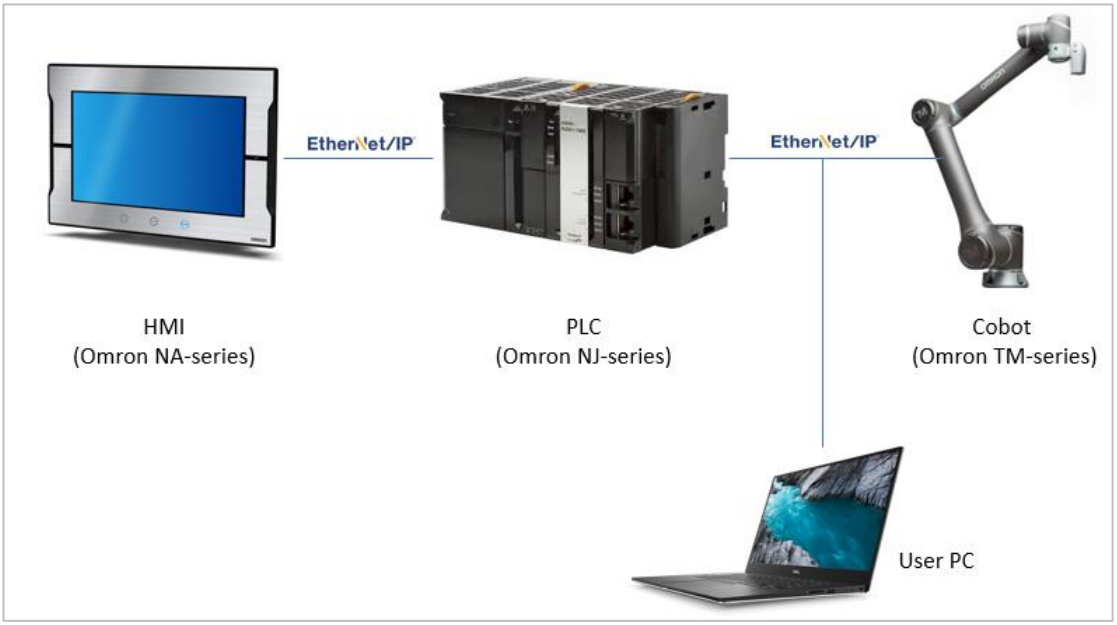

Figure 3.3.1 – Communications architectures based on Ethernet/IP (source: own).

3.4. Electrical connections

All devices are equipped with standard power supply connections. In case of the PLC, cobot, laptop, and 24Vdc power supply, they are powered with 230Vac. They can be plugged into any electrical socket available in the facility where the system will run.

Concerning HMI and Ethernet Switching Hub, 24Vdc are required. In both devices, there is a screws terminal block as power supply connector.

In the image below, the electrical drawing is a sketch designed to illustrate how the system is powered.

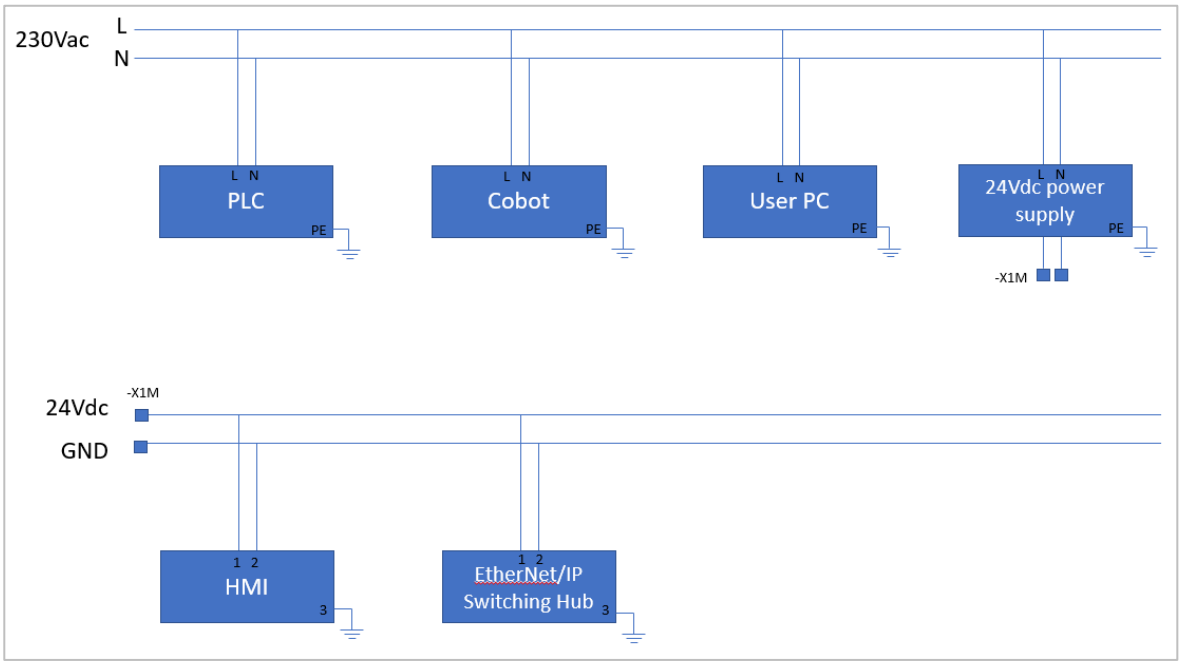

Figure 3.4.1 – Electrical connections (source: own).

4. Robot instructions

4.1. Description

Omron TM-series collaborative robot allows to establish a TCP/IP socket server connection (TCP listener) through the Listen Node function, which provides a specific protocol that allows the execution of the functions available in the Expression Editor [11] from TMflow software.

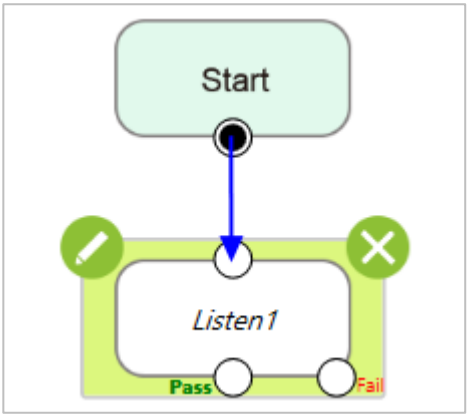

Figure 4.1.1 - In Listen Node a TCP/IP server (Socket Server) can be established and be connected by an external device (source: own).

When clicking on the pencil icon in the top-left corner of the node, a popup appears (Figure 4.1.2). In which several settings can be set:

- Node Name: identifies the node in the flowchart.
- Send message as node is entered: when the flow processed in the TMflow reaches this node, this message is sent by server (cobot) to the client (PLC).
- Print received data in log: enables the communication log on TMflow helping on debugging.
- Connection timeout: when entering in this node, if more than the time set (in milliseconds) is not connected, it will timeout. Use 0 to disable this timeout.
- Data Timeout: when entering in this node and connected, is no data is received during the time set (in milliseconds) it will timeout. Use 0 to disable this timeout.

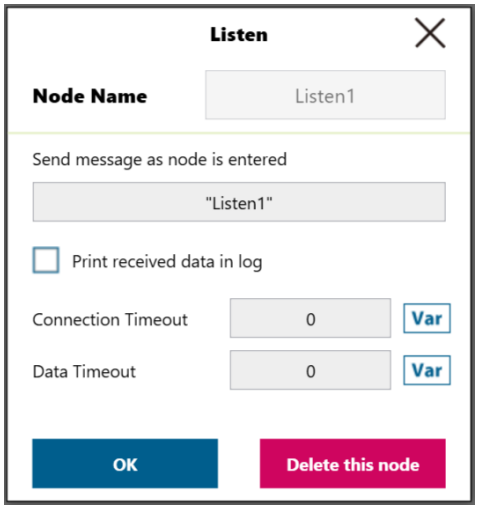

Figure 4.1.2 **-** Listen Node settings (source: own).

When the program flow execution enters in Listen Node, the flow will keep at this node until either of the two exit conditions is fulfilled:

- Pass: *ScriptExit*() command is executed and the project is stopped.
- Fail: 1. Connection Timeout.
  2. Data Timeout.

When a command is received by Listen node, it will be executed in FIFO (*First In, First Out*) order, meaning that the command is placed in the queue and executed in arrival order. In case a command is not valid, an error message is sent back to the client connection. Otherwise, if the command is valid, it will be placed on the queue.

4.2. Communications Protocol

Omron collaborative robot has its own communications protocol. Hence the commands sent from external devices must comply a very specific data structure. It is composed by data such header to identify the purpose of the message, length or quantity of bytes in the message, complete instruction including parameters, checksum to verify the message has been received correctly, and end bytes to identify the end of the message.

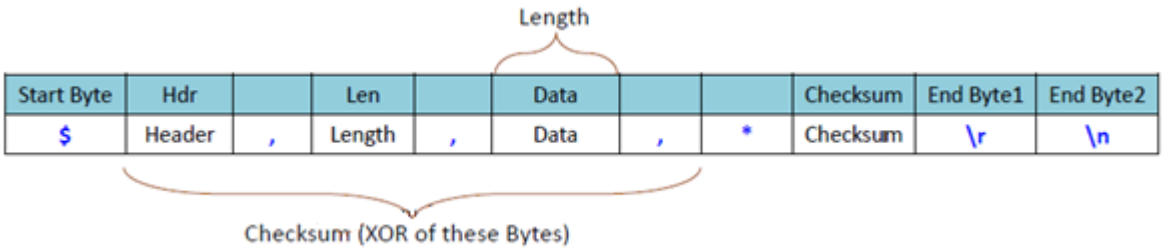

Figure 4.2.1 – Data frame structure (source: [11]).

Name	Size	ASCII	HEX	Description
Start Byte	1	$	0x24	Start Byte for Communication
Header	*X*			Header for Communication
Separator	1	,	0x2C	Separator between Header and Length
Length	*Y*			Length of Data
Separator	1	,	0x2C	Separator between Length and Data
Data	*Z*			Communication Data
Separator	1	,	0x2C	Separator between Data and Checksum
Sign	1	*	0x2A	Begin Sign of Checksum
Checksum	*2*			Checksum of Communication
End Byte 1	1	\r	0x0D	
End Byte 2	1	\n	0x0A	End Byte of Communication

Figure 4.2.2 – Data frame contents description (source: [11]).

Header

Defines the purpose of the communication package. The data contained in the command can vary depending on the Header:

- TMSCT → Function command sent from PLC.
- TMSTA → Acquiring status information or properties data:
  - Send from PLC, asking if robot is in Listen node.
  - Send from robot, returning information (answer).
- CPERR → Communication data error (e.g., package error, checksum error, header error, etc.).

Length

Defines the length in byte type. The numeric format can be decimal, hexadecimal, or binary. Example:

```
$TMSCT,100,Data,*CS\r\n      // Decimal 100, that is the data length is 100 bytes
$TMSCT,0x100,Data,*CS\r\n    // Hexadecimal 0x100, that is the data length is 256 bytes
$TMSCT,0b100,Data,*CS\r\n    // Binary 0b100, that is the data length is 4 bytes
```

Figure 4.2.3 – Length numeric format examples (source: [11]).

Data

Content of the communication data. This can vary depending on the purpose of the function command and its parameters (refer to section *4.3. Function commands*).

Checksum

The checksum of the communication package is calculated with XOR (eXclusive Or / eXclusive disjunction) logical operation. The range for the checksum calculation starts from $ and finish at * (being $ and * symbols excluded). The representation of the checksum is fixed to 2 bytes in hexadecimal format (without 0x).

```
$TMSCT,100,Data,*CS\r\n

Checksum = Byte[1] ^ Byte[2] ... ^ Byte[N-6]
```

Figure 4.2.4 – Checksum calculation (source: [11]).

Following subsection explains the three different headers accepted.

4.2.1. TMSCT

Defines the communication package as External Script Language. Use to send a command including parameters for its execution. Data contains two parts separated by comma: ID and Script.

Start Byte	Hdr		Len		Data			Checksum	End Byte1	End Byte2
$	*TMSCT*	,	Length	,	Data	,	*	Checksum	\r	\n

ID		SCRIPT
Script ID	,	Script Language

Figure 4.2.5 – Checksum calculation (source: [11]).

Data:

- ID Used as specifying the target SCRIPT of return message.
- , Separator.
- SCRIPT Multiline script containing the Script Language.

4.2.2. TMSTA

Defines the communication package to acquire status or properties. When sent from an external device, it is used to detect if the collaborative robot is in listener mode, consequently it is ready to accept and execute external commands. Data contains different subcommand (SubCMD).

Start Byte	Hdr		Len		Data			Checksum	End Byte1	End Byte2
$	*TMSTA*	,	Length	,	Data	,	*	Checksum	\r	\n

SubCmd		
SubCmd	...	...

(Based on SubCmd)

Figure 4.2.6 – Checksum calculation (source: [11]).

SubCmd:

- 00 Asks if robot is in external script control or not (Listen node).
- 01 Asks when robot motion has been completed.
- 90..99 Sends data message as variable value.

4.2.3. CPERR

Defines the communication package as Communication Error Protocol. It is used by robot to response to PLC in case an error has been detected in the received message.

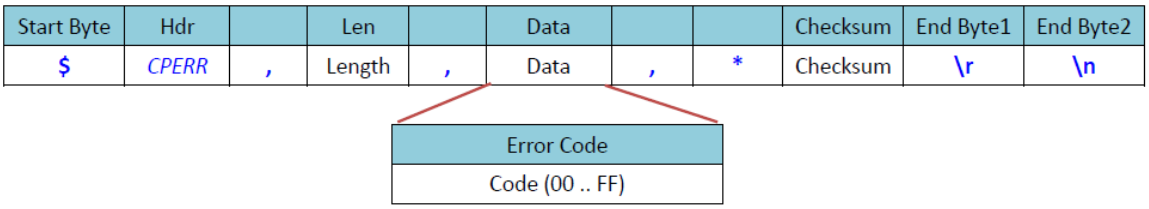

Figure 4.2.7 – Checksum calculation (source: [11]).

Error Code:

- 00 No error.
- 01 Packet error.
- 02 Checksum error.
- 03 Header error.
- 04 Packet data error.
- F1 Not in Listen Node.

4.3. Function commands

Function commands can be only performed with external scripts when the project flow is in Listen Node and it receives a message with $TMSCT as header of the command. All motion functions are queued in the buffer and executed in arrival order.

The Function Commands implemented in this library are:

- PTP()
- Line()
- Circle()
- Move_PTP()
- Move_ Line()
- ChangeBase()
- ChangeTCP()
- QueueTag()
- Pause()
- Resume()
- ScriptExit()

These functions commands [11] are described below.

4.3.1. PTP()

Define and send PTP (Point to Point) absolute motion command into buffer for execution. See description below:

Syntax

```
bool PTP(
        string,
        float, float, float, float, float, float,
        int,
        int,
        int,
        bool,
        int, int, int
)
```

Parameters

string Definition of data format, combines three letters:
#1: Motion target format:
"C" expressed in Cartesian coordinates.
#2: Speed format:
"P" expressed as a percentage (%).
#3: Blending format:
"P" expressed as a percentage (%).

float, float, float, float, float, float
Motion target location: X (mm), Y (mm), Z (mm), RX (◦), RY (◦), RZ (◦).

int Speed setting expressed as a percentage (%).
int Time interval to accelerate to top speed (ms).
int Blending value expressed as a percentage (%).

bool Disable precise positioning.
true Disable precise positioning.
false Enable precise positioning.

int, int, int
Pose of robot: Config1, Config2, Config3.

- Config1:
  - RIGHTY = 0
  - LEFTY = 1

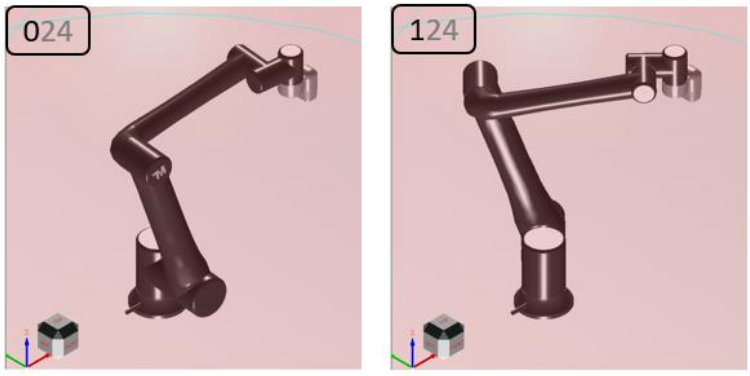

Figure 4.3.1 – Config1 pose of robot: RIGHTY = 0, LEFTY = 1 (source: own).

- Config2:
  - ABOVE = 2
  - BELOW = 3

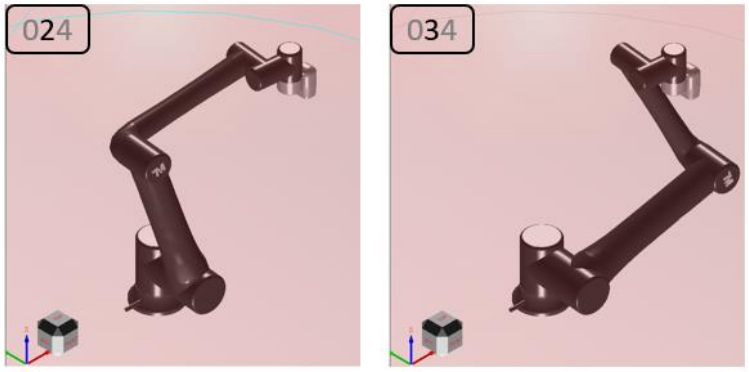

Figure 4.3.2 – Config2 pose of robot: ABOVE = 2, BELOW = 3 (source: own).

- Config3:
  - NOFLIP = 4
  - FLIP = 5

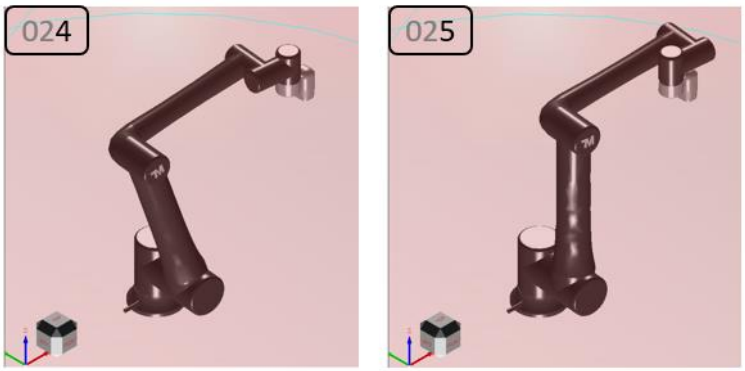

Figure 4.3.3 – Config3 pose of robot: NOFLIP = 4, FLIP = 5 (source: own).

Return

bool *True* Command accepted; *False* Command rejected (format error).

Note

Data format parameter includes: (1) "CPP".

8 different robot pose configurations are possible:

Figure 4.3.4 – Pose of robot can be in 8 different configurations: 024: RIGHTY, ABOVE, NOFLIP; 025: RIGHTY, ABOVE, FLIP; 034: RIGHTY, BELOW, NOFLIP; 035: RIGHTY, BELOW, FLIP; 124: LEFTY, ABOVE, NOFLIP; 125: LEFTY, ABOVE, FLIP; 134: LEFTY, BELOW, NOFLIP; 135: LEFTY, BELOW, FLIP (source: own).

Example

```
PTP("CPP",417.50,-122.30,343.90,180.00,0.00,90.00,10,200,0,false,0,2,4)
//Move to coordinate (417.50, -122.30, 343.90, 180.00, 0.00, 90.00), with PTP, speed=10%,
time to top speed=200ms, no blending, precise positioning disabled, pose = 024)
```

4.3.2. Line()

Define and send Line absolute motion command into buffer for execution. See description below:

Syntax

```
bool Line(
        string,
        float, float, float, float, float, float,
        int,
        int,
        int,
        bool
)
```

Parameters

string Definition of data format, combines three letters:
- #1: Motion target format:
  - "C" expressed in Cartesian coordinates.
- #2: Speed format:
  - "P" expressed as a percentage (%).
  - "A" expressed in velocity (mm/s).
- #3: Blending format:
  - "P" expressed as a percentage (%).
  - "R" expressed in radius (mm).

float, float, float, float, float, float
Motion target location: X (mm), Y (mm), Z (mm), RX (◦), RY (◦), RZ (◦).

int Speed setting expressed as a percentage (%) or in velocity (mm/s).
int Time interval to accelerate to top speed (ms).
int Blending value expressed as a percentage (%) or in radius (mm).

bool Disable precise positioning.
- *true* Disable precise positioning.
- *false* Enable precise positioning.

Return

bool *True* Command accepted; *False* Command rejected (format error).

Note

Data format parameter includes: (1) "CPP", (2) "CPR", (3) "CAP" and (4) "CAR".

Example

Line("CAR",417.50,-122.30,343.90,180.00,0.00,90.00,100,200,50,false)
//Move to coordinate (417.50, -122.30, 343.90, 180.00, 0.00, 90.00), with Line, speed=100mm/s, time to top speed=200ms, blending radius = 50mm and precise positioning disabled.

4.3.3. Circle()

Define and send Circle absolute motion command into buffer for execution. See description below:

Syntax

```
bool Circle(
        string,
        float, float, float, float, float, float,
        float, float, float, float, float, float,
        int,
        int,
        int,
        int,
        bool
)
```

Parameters

string Definition of data format, combines three letters:
- #1: Motion target format:
  - "C" expressed in Cartesian coordinates.
- #2: Speed format:
  - "P" expressed as a percentage (%).
  - "A" expressed in velocity (mm/s).
- #3: Blending format:
  - "P" expressed as a percentage (%).

float, float, float, float, float, float
A point on arc: X (mm), Y (mm), Z (mm), RX (◦), RY (◦), RZ (◦).

float, float, float, float, float, float
The end point of arc: X (mm), Y (mm), Z (mm), RX (◦), RY (◦), RZ (◦).

int Speed setting expressed as a percentage (%) or in velocity (mm/s).

int Time interval to accelerate to top speed (ms).

int Blending value expressed as a percentage (%).

Int Arc angle (◦). If non-zero value is given, the TCP will keep the same pose and move from current point to the assigned arc angle via the given point and end point on arc. If zero is given, the TCP will move from current point and pose to end point and pose via the point on arc with linear interpolation on pose.

bool Disable precise positioning.
- *true* Disable precise positioning.
- *false* Enable precise positioning.

Return

bool *True* Command accepted; *False* Command rejected (format error).

Note

Data format parameter includes: (1) "CPP", (2) "CAP".

Example

```
Circle("CAP",417.50,-122.30,343.90,180.00,0.00,90.00,
381.70,208.74,343.90,180.00,0.00,135.00,100,200,50,270,false)
//Via point = (417.50, -122.30, 343.90, 180.00, 0.00, 90.00), end point = (381.70, 208.74, 343.90, 180.00, 0.00, 135.00), move on 270 degrees arc, speed=100mm/s, time to top speed=200ms, blending value radius = 50%, precise positioning disabled.
```

4.3.4. Move_PTP()

Define and send PTP relative motion command into buffer for execution. See description below:

Syntax

```
bool Move_PTP(
        string,
        float, float, float, float, float, float,
        int,
        int,
        int,
        bool
)
```

Parameters

string Definition of data format, combines three letters:

#1: Relative motion target format:

"C" expressed in Current base coordinates.

"T" expressed in Tool coordinates.

"J" expressed in Joint angle.

#2: Speed format:

"P" expressed as a percentage (%).

#3: Blending format:

"P" expressed as a percentage (%).

float, float, float, float, float, float
Relative motion target location. If expressed in current base coordinates or tool coordinates, it includes the tool end TCP relative motion value with respect to the specified coordinate: X (mm), Y (mm), Z (mm), RX (◦), RY (◦), RZ (◦). If defined with joint angle, it includes the angles of six joints: Joint1 (◦), Joint2 (◦), Joint3 (◦), Joint4 (◦), Joint5 (◦), Joint6 (◦).

int Speed setting expressed as a percentage (%).
int Time interval to accelerate to top speed (ms).
int Blending value expressed as a percentage (%).

bool Disable precise positioning.
true Disable precise positioning.
false Enable precise positioning.

Return

bool *True* Command accepted; *False* Command rejected (format error).

Note

Data format parameter includes: (1) "CPP", (2) "TPP", (3) "JPP".

Example

```
Move_PTP("TPP",0,0,10,45,0,0,10,20,0,false)
//Move (0, 0, 10, 45, 0, 0) with respect to tool coordinate, with PTP motion, speed =10%, time to top speed = 200ms, no blending, precise positioning disabled.
```

4.3.5. Move_Line()

Define and send Line relative motion command into buffer for execution. See description below:

Syntax

```
bool Move_Line(
    string,
    float, float, float, float, float, float,
    int,
    int,
    int,
    bool
)
```

Parameters

string Definition of data format, combines three letters:
- #1: Relative motion target format:
  - "C" expressed in Current base coordinates.
  - "T" expressed in Tool coordinates.
- #2: Speed format:
  - "P" expressed as a percentage (%).
  - "A" expressed in velocity (mm/s).
- #3: Blending format:
  - "P" expressed as a percentage (%).
  - "R" expressed in radius (mm).

float, float, float, float, float, float
Relative motion target. It includes the tool end TCP relative motion value with respect to the specified current base or tool coordinate: X (mm), Y (mm), Z (mm), RX (◦), RY (◦), RZ (◦).

int Speed setting expressed as a percentage (%) or in velocity (mm/s).
int Time interval to accelerate to top speed (ms).
int Blending value expressed as a percentage (%) or in radius (mm).

bool Disable precise positioning.
- *true* Disable precise positioning.
- *false* Enable precise positioning.

Return

bool *True* Command accepted; *False* Command rejected (format error).

Note

Data format parameter includes: (1) "CPP", (2) "CPR", (3) "CAP", (4) "CAR", (5) "TPP", (6) "TPR", (7) "TAP" and (8) "TAR".

Example

Move_Line("TPP",0,0,10,45,0,0,10,20,0,false)
//Move (0, 0, 10, 45, 0, 0) with respect to tool coordinate with Line motion, speed =10%, time to top speed = 200ms, no blending, precise positioning disabled.

4.3.6. ChangeBase()

Define and send the command of changing the base of the follow-up motions into buffer for execution. See description below:

Syntax

```
bool ChangeBase(
        float, float, float, float, float, float
)
```

Parameters

float, float, float, float, float, float

Base parameters combining X (mm), Y (mm), Z (mm), RX (◦), RY (◦), RZ (◦).

Return

bool *True* Command accepted; *False* Command rejected (format error).

Example

```
ChangeBase(20,30,10,0,0,90)
//Change the base value to (20, 30, 10, 0, 0, 90).
```

4.3.7. ChangeTCP()

Define and send the command of changing the TCP of the follow-up motions into buffer for execution. See description below:

Syntax

```
bool ChangeBase(
        float, float, float, float, float, float,
        float,
        float, float, float, float, float, float, float, float, float
)
```

Parameters

float, float, float, float, float, float

TCP parameters combining X (mm), Y (mm), Z (mm), RX (◦), RY (◦), RZ (◦).

float

Tool's weight.

float, float, float, float, float, float, float, float, float

Tool's moment of inertia: (1)Ixx (kg·mm²), (2)Iyy (kg·mm²), (3)Izz (kg·mm²) and its frame of reference (4)X (mm), (5)Y (mm), (6)Z (mm), (7)RX (◦), (8)RY (◦), (9)RZ (◦).

Return

bool *True* Command accepted; *False* Command rejected (format error).

Example

```
ChangeTCP(0,0,150,0,0,90,2,2,0.5,0.5,0,0,-80,0,0,0)
//Change the TCP value to (0, 0, 150, 0, 0, 90), weight = 2kg, moment of inertia = (2, 0.5
,0.5) , and frame of reference = (0, 0, -80, 0, 0, 0).
```

4.3.8. QueueTag()

Identify the robot motion with a Number to return feedback when robot motion finishes the current robot motion in process. See description below:

Syntax

```
bool QueueTag(
    int,
    int
)
```

Parameters

int

The tag number. Valid for integers between 1 and 15.

int

Wait for the tagging to continue processing or not:
0 Not wait.
1 Wait.
When the value is set to 0, no wait for tagging to continue processing.
When the value is set to 1, the process stays in the function and waits for the tagging to complete and continue processing.

Return

bool *True* When tagged successfully; *False* when tagged unsuccessfully.

Example

```
QueueTag(8,0)
//Tags the robot motion with number 8 for returning feedback when robot motion finishes
```

4.3.9. Pause()

Pause the project and the motion of the robot. See description below:

Syntax

```
bool Pause(
)
```

Parameters

void
No input values required.

Return

bool *True* Command accepted; *False* Command rejected (format error).

Example

```
Pause()
```

4.3.10. Resume()

Resume the project and the motion of the robot. See description below:

Syntax

bool Resume(
)

Parameters

void
No input values required.

Return

bool *True* Command accepted; *False* Command rejected (format error).

Example

Resume()

4.3.11. ScriptExit()

Exit external script control mode once the last motion command has been executed and finished. Listen Node is quitted, and not further external script functions commands are accepted. See description below:

Syntax

bool ScriptExit(
)

Parameters

void
No input values required.

Return

bool *True* Command accepted; *False* Command rejected (format error).

Note

Exit the external script control mode and wait for the command to finish, and then quit the listen node and move on with the pass route.

Example

ScriptExit()

5. Library

This library encapsulates many of the Expression Editor [11] functions from TMflow. Among them, the most relevant ones in terms of robot motion, coordinate parameters and program execution control. The seven Function Blocks developed for this library are:

- CBT_ChangeBase
- CBT_ChangeTCP
- CBT_Connect
- CBT_MoveAbsolute
- CBT_MoveRelative
- CBT_MoveCircle
- CBT_ProgramControl

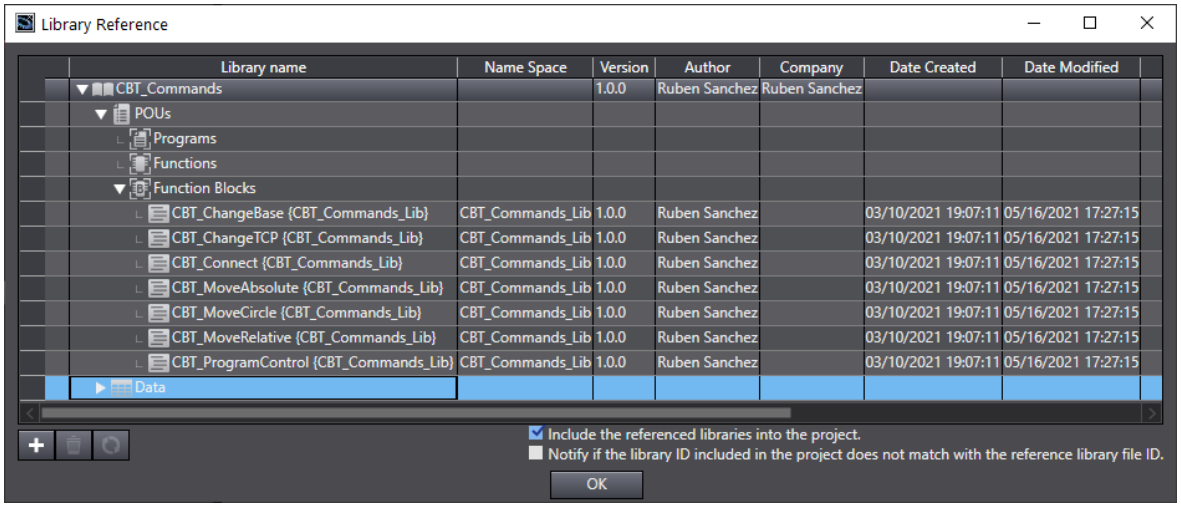

Figure 5.0.1 – Library reference when imported to the PLC project (source: own).

In some cases, different functions have been included inside the same FB. *PTP()* and *Line()* functions have been encapsulated in *CBT_MoveAbsolute*; or *Move_PTP()* and *Move_Line()* functions have been encapsulated in *CBT_MoveRelative*. In both cases, there is an input parameter in which 'Line' or 'PTP' option can be selected in order to execute one or another.

Once the customer wants to use the library, it must be loaded into the software to be used in the project in which the PLC is programmed. When user enters in programming editor in Sysmac Studio environment, all FBs in the library are displayed in the Toolbox (Figure 5.0.2), thus customer can drag & drop the FBs into the program.

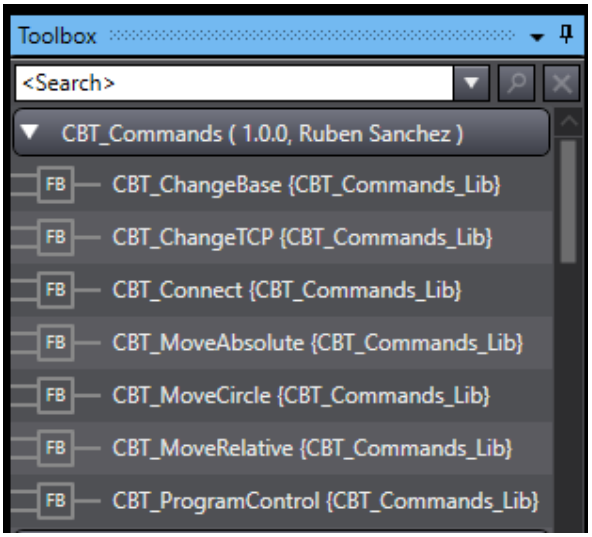

Figure 5.0.2 – Library shown in the toolbox of Sysmac Studio once it has been imported onto the PLC project (source: own).

5.1. PLCopen and IEC 61131-3 compliant Function Blocks

PLCopen [1] is an independent organization, founded in 1992, providing support to the user community in terms of harmonization of control programming and application, and interfacing engineering for industrial automation.

IEC 61131-3 standard [4] for PLC is the unified suite of programming languages for programmable controllers. Programming languages like:

- Ladder diagram (LD).
- Function block diagram (FBD).
- Structured text (ST).
- Instruction list (IL).
- Sequential function chart (SFC).

PLCopen and IEC 61131-3 provide the basis to achieve higher efficiency during application development, maintenance over the life cycle, adding new functionalities, etc. This is the reason why it has been decided to follow their guidelines.

5.1.1. Function Block appearance

The document "Creating PLCopen compliant Function Block libraries" [3] has been a great help to understand how a Function Block must behave and how its appearance should be. The behaviour of the FBs has been described in section 5.1.2).

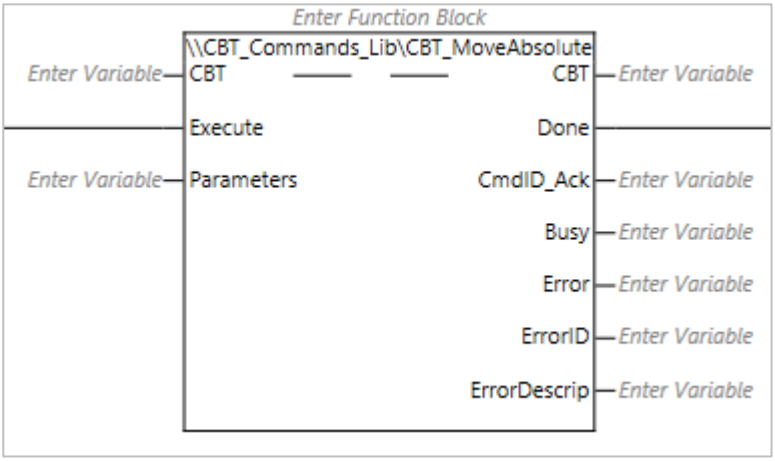

Figure 5.1.1 – Function block appearance similar for all FBs in the library (source: own).

As one of the programming languages stated in IEC 61131-3 standard, the algorithm inside each Function Block has developed in Structured text (ST) programming language.

```
10: (*  Initialization  *)

    TCP_Clear_Buffer_Exe :=FALSE;
    TCP_Send_Exe         :=FALSE;
    TCP_Rcv_Exe          :=FALSE;
    ToErrorID            :=0;
    ToErrorDescrip       :=' ';
    CmdID_Ack            :=Parameters.CommandID;

    IF CBT.Connected and NOT(CBT.WaitingReturn) THEN    //If robot is connected and "released" (not occupied by other FB)
       FB_step:=20;
    ELSIF NOT(CBT.Connected) THEN                       //Robot not connected
       ToError_Sts := TRUE;
       ToErrorID:=16#10;
       ToErrorDescrip:='Robot not connected';
    ELSIF (CBT.WaitingReturn) THEN                      //Robot is "occupied" by other FB
       ToError_Sts := TRUE;
       ToErrorID:=16#11;
       ToErrorDescrip:='Other FB is under execution';
    END_IF;
```

Figure 5.1.2 – Part of the algorithm of a FB from the library programmed in Structured Text (source: own).

The script in each Function block has been implemented following a methodological principle in computations software engineering known as SoC (Separation-Of-Concerns) [5]. This principle states that a program design must be separated in distinct sections, one per concern.

Separation-Of-Concerns has been implemented in the library at two different levels. In terms of library contents, there is not only 1 FB doing all actions, there are 7 different FBs and each one has its own purpose, but conceptually, they can be classified in 4 groups:

- Communications connection (CBT_Connect).
- Robot movements (CBT_MoveAbsolute, CBT_MoveRelative, CBT_MoveCircle).
- Coordinates functions (ChangeBase, ChangeTCP).
- Program control (CBT_ProgramControl).

The second implementation of Separation-Of-Concerns is in terms of scripting structure inside each FB. There are 2 main and separated sequences that are linked each other but they have different purposes:

- "*State Diagram Control*" focused on monitoring execution state. It updates the FB outputs (Done, Busy, Error, etc.) depending on the FB inputs and the *Algorithm Sequence*.
- "*Algorithm Sequence*" interprets the data provided in the Parameters input, converts Parameters into strings, builds the TCP/IP frame, and sends the command to the robot.

Both sequences are described in the following sections.

5.1.2. State Diagram Control

State Diagram Control is the part of the script focused on output variable operation and timing. It is responsible of how the algorithm behaves depending on the input variables and how the output variables monitor the status of the internal algorithm. The common IO variables are:

- Execute: input variable that gives the execution condition for the FB.
- Done: output variable that shows the completion of the execution for the FB.
- Busy: output variable that shows execution in progress for the FB.
- Error: output variable that indicates error end flag for the FB.
- ErrorID: output variable that indicates error number (hexadecimal) for the FB.
- ErrorDescription: output variable that indicates error description for the FB.

Values of the output variables can be monitored to determine the status throughout instruction execution. Execution starts when *Execute* input changes to TRUE, then *Busy* output changes to TRUE, *Done* output changes to FALSE, and *Error* output changes to FALSE.

In case of "Normal End": *Busy* changes to FALSE and *Done* changes to TRUE. *Error* does not change (remains FALSE).

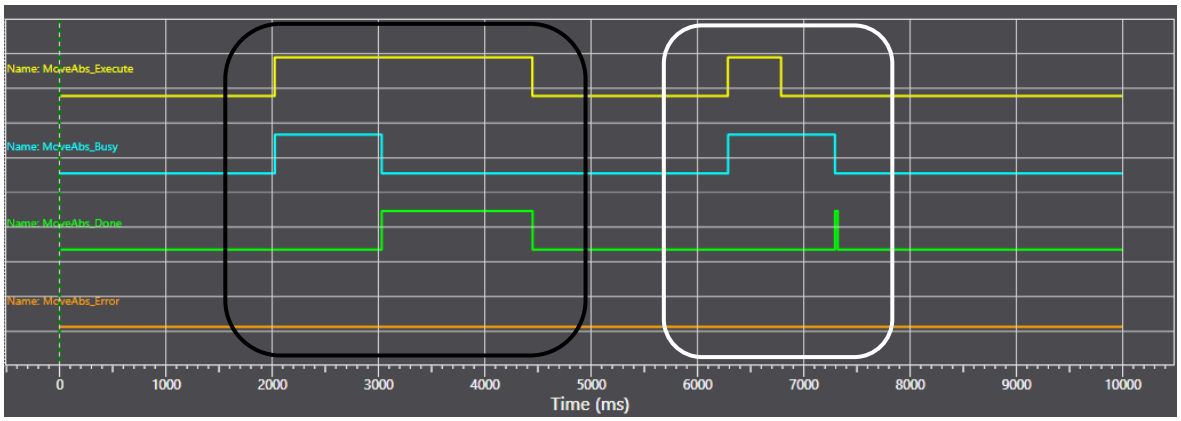

Figure 5.1.3 – Normal end output variable operation and timing chart for *Execute* (yellow), *Busy* (blue), *Done* (green), *Error* (orange). Black square highlights the example if *Execute* is TRUE until *Done* changes to TRUE. *Done* stays TRUE until *Execute* changes to FALSE. White square highlights the example if *Execute* changes back to FALSE before *Done* changes to TRUE. *Done* stays TRUE for only one task period (source: own).

In case of "Error End": *Busy* changes to FALSE and *Error* changes to TRUE. *Done* does not change (remains FALSE). *ErrorID* and *ErrorDescription* display valuable information for error identification.

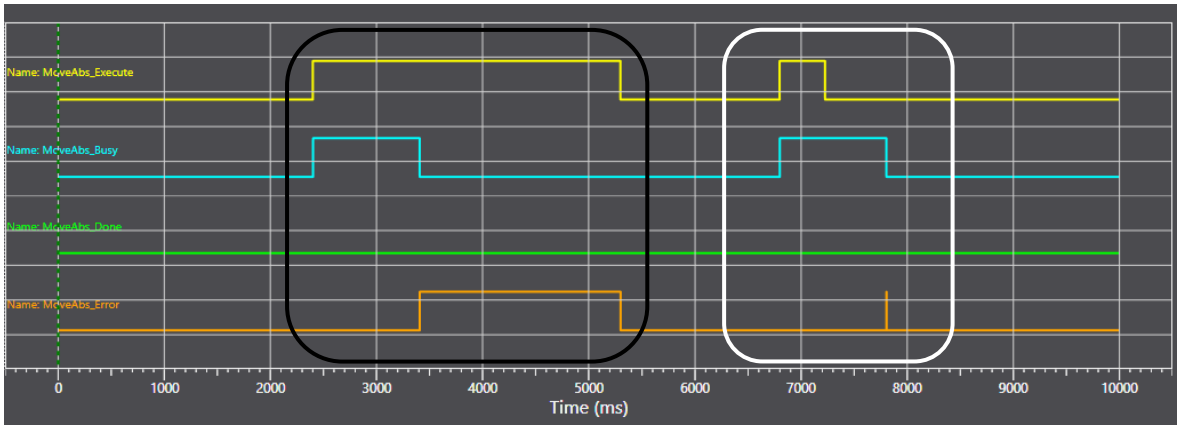

Figure 5.1.4 – Normal end output variable operation and timing chart for *Execute* (yellow), *Busy* (blue), *Done* (green), *Error* (orange). Black square highlights the example if *Execute* is TRUE until *Error* changes to TRUE. *Error* stays TRUE until *Execute* changes to FALSE. White square highlights the example if *Execute* changes back to FALSE before *Error* changes to TRUE. *Error* stays TRUE for only one task period. Error Description output remains showing the message until FB is executed again (source: own).

For those Function Blocks related to robot motion (CBT_MoveAbsolute, CBT_MoveRelative, CBT_MoveCircle) *Done* output behaves different depending on the *EnableBlending* input parameter:

- If *EnableBlending* is TRUE, *Done* changes to TRUE when command has been accepted by the robot even though the movement has not been completed yet. It allows to send next motion command and put it in the robot queue before current movement is completed, thus blending between movements can be applied by the robot.
- If *EnableBlending* is FALSE, *Done* changes to TRUE when the robot movement has been completed. Useful in those cases in which the robot should do actions like open/close the gripper or activate a signal once the robot has achieved a certain position.

5.1.3. Algorithm Sequence

The algorithm structure inside the Function Blocks is similar each other: When *Execute* input changes to TRUE, if the FB is not being executed at that moment, the values of all input parameters are saved in internal variables. The reason is preventing the values are not changed during FB execution. Then, if the robot is connected to the PLC and, after clearing the communications port buffer, it is verified that all values of input parameters are inside the acceptable range. Next step is converting all input parameters into string format to create the complete message. Once the number of characters is found, the checksum is obtained (these concepts are described in detail in section *5.3. Function Blocks description*). Finally, the message is sent to the robot and the FB waits until the return message is received from robot. Thus, the FB execution is finished.

The sequence verifies on each step if the operation is valid in order to proceed with the step after. Otherwise, if operation is not valid or an error exists, the FB will trigger the error output and will "Error end". Therefore, *ErrorID* and *ErrorDescrip* provides valuable information helping to identify and debug the issue.

In general terms, all Function Blocks follow the same sequence: get data from input parameters, build the frame, send the frame to the robot, wait for response and finish execution (see Figure 5.1.3.1).

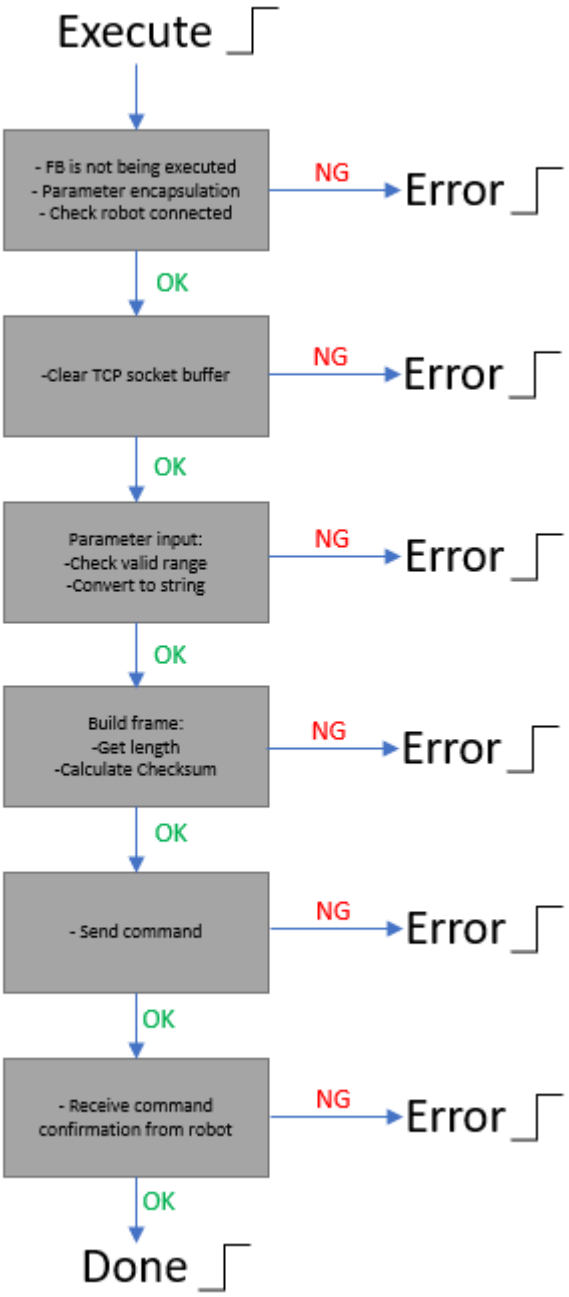

Figure 5.1.5 – Flow chart structure for all FBs in the library. On each step (square), and before executing the next step, there is a check: if there is an error or the operation is not valid, the FB triggers "Error End". Each arrow represents where the flow goes depending on the result: *NG* means No Good, *OK* means continue with next step (source: own).

5.2. Library structure

5.2.1. Namespace and Datatypes

In the library, Function Blocks and Datatypes have been nested in Namespaces allowing to group the name of the FB and datatype definition to manage them reducing the chance of duplicated names and making the entities easier to access. Following two images show the notation for a name nested in the namespace (*CBT_Commands_Lib*) created for this library:

Figure 5.2.1 – Notation for a name that uses a namespace (source: own).

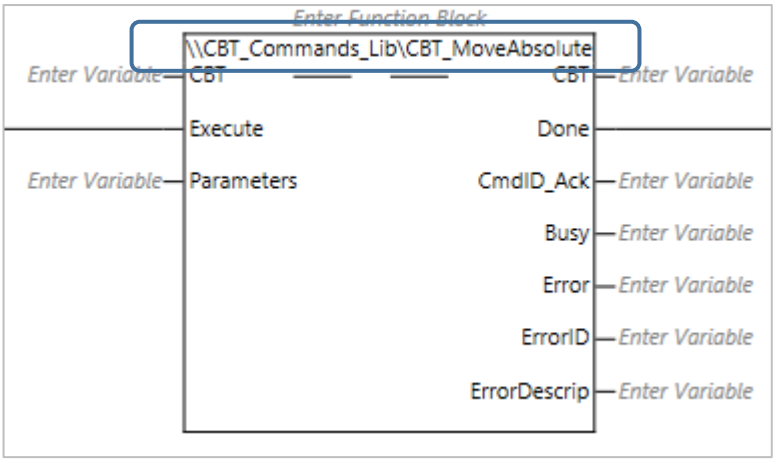

Figure 5.2.2 – Example of FB nested in a namespace with corresponding notation (source: own).

There are 2 main datatypes defined in this library. The prefix of their name identifies which kind of datatype they are:

- Datatypes starting with “stCBT_” are structure type.
- Datatypes starting with “eCBT_” are enumerated type.

Name	Base Type
▼ stCBT_MovRelParam	STRUCT
CommandID	UINT
MovementCommand	CBT_Commands_Lib\eCBT_MovCmd_Movement
DataFormat	CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat
TargetPosition	CBT_Commands_Lib\stTransformation
Speed	UINT
AccelTime	UINT
BlendingEnable	BOOL
BlendingValue	UINT
PrecisePositioning	BOOL
ExitNode	BOOL

Figure 5.2.3 – Example of Structure datatype nested in CBT_Commands_Lib namespace (source: own).

Name	Enum Value	Comment
▼ eCBT_MovCmd_Movement		
Line	0	Motion in Joint, speed in %, blending in %
PTP	1	Motion in Cartesian, speed in %, blending in %

Figure 5.2.4 – Example of Enumerated datatype nested in CBT_Commands_Lib namespace (source: own).

5.2.2. Function Block description

There are two types of variables for the Function Block: One type is system-defined variable, used to monitor the robot status and parameters (*In-out variables*). The second type variable are used to input arguments (*Input variables*) or to output execution status (*Output variables*). Some input variables are enumerated type which selection are made from a set of predefined enumerators.

The three variables for FB instructions are:

- *In-out variables* specify data to process with the instruction.
- *Input variables* are instruction arguments.
- *Output variables* are instruction execution status monitoring information.

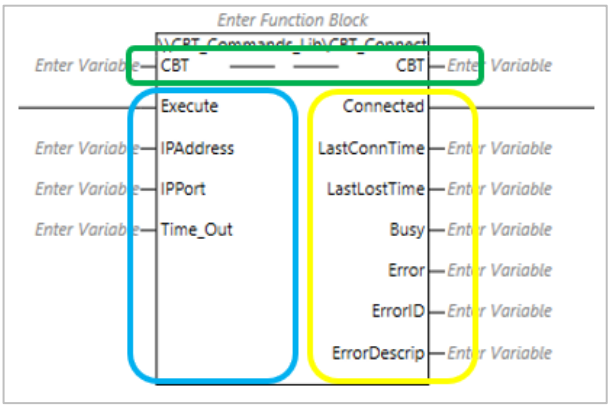

Figure 5.2.5 – System-defined variable/In-out variable (green), arguments/input variables (blue) and output status/output variables (yellow) (source: own).

5.3. Function Blocks description

Hereunder there is the description of each FB in the library. The first FB is described completely. The second one is described partially, just what is different from first FB. The FBs after are described partially, just what is different from previous FBs.

Each FB has its own purpose, but they can be classified by concept: Communications connection, robot motion, coordinate parameters and program execution control. The seven Function Blocks developed for this library are:

- CBT_Connect
- CBT_MoveAbsolute
- CBT_MoveRelative
- CBT_MoveCircle
- CBT_ChangeBase
- CBT_ChangeTCP
- CBT_ProgramControl

5.3.1. CBT_Connect

This Function Block allows to open a connection between the PLC and the collaborative robot. It also supervises the connection all the time it is in execution by intervals of 1 second:

- If robot program in TMflow is not running when *Execute* input changes to TRUE, the FB is cyclically trying to connect with the robot until robot program in TMflow runs (robot connected) or timeout is exceeded (Error End).
- If robot program in TMflow stops when connection is stablished (the FB execution is in progress), robot will close the connection (Error End).
- If *Execute* input changes to FALSE, PLC is disconnected, and connection is closed (Normal End).
- Meanwhile PLC is connected to robot, the *Connected* output remains TRUE, otherwise it changes to FALSE.
- Once the connection becomes effective, *LastConnTime* output shows date and time when the connection was done.
- Immediately after the connection breaks down, *LastLostTime* output shows date and time when the connection was lost. In *ErrorDescrip* output, the reason of the disconnection can be known.

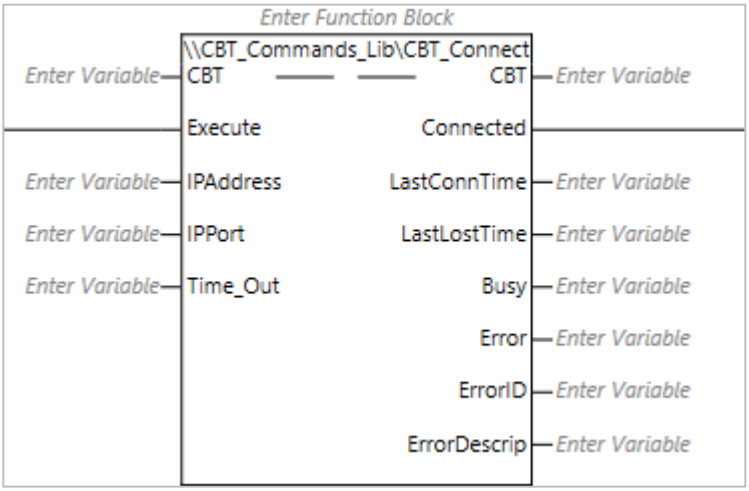

Figure 5.3.1.1 – CBT_Connect Function Block (source: own).

In the below tables, input and output parameters are described:

Name	I/O	Data Type	Valid Range	Default	Description
CBT	In-Out	CBT_Commands_Lib\stCBT_Connection	---	---	Identifies robot IP address, port and connection.
Execute	Input	BOOL	TRUE or FALSE	FALSE	Request of instruction execution
IPAddress	Input	STRING[50]	---	---	Destination IP address
IPPort	Input	UINT	0 to +65535	0	Destination TCP port number
Time_Out	Input	UINT	10 to +300	0	Time in seconds in which the FB is trying to connect
Connected	Output	BOOL	TRUE or FALSE	FALSE	TRUE when the cobot is connected
LastConnTime	Output	DATE_AND_TIME	---	---	Date time of last connection stablished
LastLostTime	Output	DATE_AND_TIME	---	---	Date time of last connection lost
Busy	Output	BOOL	TRUE or FALSE	FALSE	TRUE when the instruction is in progress
Error	Output	BOOL	TRUE or FALSE	FALSE	TRUE while there is an error
ErrorID	Output	WORD	---	16#0000	Contains the error code when an error occurs. A value of 16#0000 indicates normal execution
ErrorDescrip	Output	STRING[50]	---	---	Contains the error description when an error occurs

Table 5.3.1.1 – CBT_Connect variables type, range, default value and description (source: own).

The variable name *stCBT_Connection* is an in-out structure variable common in all Function Blocks in the Library. *stCBT_Connection* identifies the robot IP address and robot IP port, monitors if the robot is connected and if it is commanded by any Function Block in the library.

Name	Data Type	Valid Range	Default	Description
Socket	_sSOCKET	---	---	Socket
Connected	BOOL	TRUE or FALSE	FALSE	TRUE when robot is connected
WaitingReturn	BOOL	TRUE or FALSE	FALSE	TRUE when robot is under control

Table 5.3.1.2 – stCBT_Connection variables type, range, default value and description (source: own).

CBT_Connect FB requests a connection between local TCP port number *SrcAdr.PortNo* and destination TCP port number *DstAdr.PortNo* at destination address *DstAdr.IpAdr*. Settings contained in the datatype of Socket which structure is *_sSOCKET* (specifications are as shown in Table 5.3.1.3).

Connected signal is TRUE when connection is done. The main purpose is to identify if the robot program is in Listen Node allowing to the FBs to send the command to the robot. *Connected* signal is FALSE if the robot program is not in the Listen Node thus robot cannot execute any command from the external device.

WaitingReturn signal is TRUE when a FB has sent a command to the robot and the FB is waiting to return message from robot program. Once the message is received by the PLC, *WaitingReturn* changes to FALSE.

Name	I/O	Data Type	Valid Range	Default	Description
Socket		_sSOCKET	---	---	Socket
Handle		UDINT	Depends on data type	0	Handle for data communications
SrcAdr		_sSOCKET_ADDRESS	---	---	Local IP address and port number
SrcAdr.PortNo		UDINT	1 to 65535	0	Port number
SrcAdr.IpAdr		STRING	Depends on data type	''	Source IP address
DstAdr		_sSOCKET_ADDRESS	---	---	Local IP address and port number
DstAdr.PortNo		UDINT	1 to 65535	0	Port number
DstAdr.IpAdr		STRING	Depends on data type	''	Destination IP address

Table 5.3.1.3 – *_sSOCKET* variable structure type, range, default value and description (source: own).

Function Block Script

Once the upward signal differentiation of *Execute* input is detected, the sequence initialization is done:

- If the FB status is different than 2 (this is no *Busy*) then, initiates the State Diagram Control (FB_status:=1). If FB is in Busy state, it omits the re-execution.
- Initiates the Algorithm Sequence (FB_Step:=10).
- Encapsulates the input parameters into internal variables avoiding the user to change those values during FB execution.

```
16  // Detect rising flag on Execute input
17  R_TRIG_Execute(Clk:=Execute, Q=>flagExecute);
18
19  //Sequence initialization
20  IF flagExecute THEN
21      IF FB_status <> 2 THEN
22          //initates secuences
23          FB_status:=1;
24          FB_step:=10;
25
26          //Save input at FB Execute to avoid input values modification during FB execution
27          Local_PortIP    := 0;           // Local TCP port number: Automatically assigned.
28          AddressIP       := IPAddress;   // Cobot IP address
29          Dest_PortIP     := IPPort;      // Cobot IP port
30      END_IF;
31  END_IF;
```

Figure 5.3.1.2 – Sequence initialization similar for all FBs (source: own).

Any time in which *Execute* signal becomes FALSE, Algorithm Sequence jumps to Request Closing step (FB_Step:=100). Therefore the connection is closed and FB is finished.

```
32
33  //Close connection
34  F_TRIG_Execute(Clk:=Execute, Q=>FalseExecute);
35  IF FalseExecute THEN
36      FB_step:=100;
37  END_IF;
```

Figure 5.3.1.3 – Close connection common for all FBs (source: own).

State Diagram Control is part of the script focused on output variable operation and timing. It is compoused by 7 diferent steps:

```
40  (* FB State Diagram Control                                                    *)
41  (* ---------------------------------------------------------------------------*)
42    //FBstatus_Step :  0 - Idle
43    //FBstatus_Step :  1 - Reset
44    //FBstatus_Step :  2 - Busy
45    //FBstatus_Step :  3 - Error
46    //FBstatus_Step :  4 - Error deactive
47    //FBstatus_Step :  5 - Done active
48    //FBstatus_Step :  6 - Done deactive
```

Figure 5.3.1.4 – FB State Diagram Control steps common for all FBs (source: own).

The Function Block remains in step 0 when it is idle or not in usage.

```
51  //Sequence
52 CASE FB_status OF
53
54 0: (*    Idle        *)
55      FB_status:=0;
56
```

Figure 5.3.1.5 – FB State Diagram Control step 0 common for all FBs (source: own).

Step 1 initializes all outputs: Boolean are FALSE, Numeric are equal to 0 and String variables are emptied.

```
57 1: (*    Reset       *)
58      ToError_Sts:=FALSE;
59      ToDone_Sts:=FALSE;
60
61      Done := FALSE;
62      Busy := FALSE;
63      Error := FALSE;
64      ErrorID:=0;
65      ErrorDescrip:=' ';
66      FB_status:=2;
67
```

Figure 5.3.1.6 – FB State Diagram Control step 1 common for all FBs (source: own).

The Function Block remains in step 2 meanwhile it is under execution (*Busy* state). It waits for *Error* flag of *Done* flag from Algorithm Sequence to change its step and refresh output signals.

```
68 2: (* Busy state   *)
69      Done := FALSE;
70      Busy := TRUE;
71      Error := FALSE;
72      ErrorID:=0;
73      ErrorDescrip:=' ';
74
75      IF ToError_Sts THEN
76         FB_status:=3;      //Error flag
77      END_IF;
78
79      IF ToDone_Sts THEN
80         FB_status:=5;      //Done flag
81      END_IF;
82
```

Figure 5.3.1.7 – FB State Diagram Control step 2 common for all FBs (source: own).

Whether an error occurs, then it moves from Step 2 to Step 3 in which *Busy* changes to FALSE and *Error* signal is TRUE. Once *Execute* signal is FALSE, it moves to Step 4. This ensures *Error* signal is TRUE at least for 1 PLC scan cycle.

```
82
83  3: (* Error state   *)
84      Done := FALSE;
85      Busy := FALSE;
86      Error := TRUE;
87      ErrorID:=ToErrorID;
88      ErrorDescrip:=ToErrorDescrip;
89
90      IF NOT Execute THEN
91          FB_status:= 4;          //Returns to idle
92      END_IF;
```

Figure 5.3.1.8 – FB State Diagram Control step 3 common for all FBs (source: own).

In Step 4, all boolean signals are set to FALSE, only *ErrorDescription* output monitors the information from the last error triggered.

```
93
94  4: (* Error Deactive if not execute   *)
95      Done := FALSE;
96      Busy := FALSE;
97      Error := FALSE;
98      ErrorID:=ToErrorID;
99      ErrorDescrip:=ToErrorDescrip;
100
```

Figure 5.3.1.9 – FB State Diagram Control step 4 common for all FBs (source: own).

Otherwise, if execution is finished correctly, then it moves from Step 2 to Step 5 in which *Busy* changes to FALSE and *Done* signal is TRUE. Once *Execute* signal is set to FALSE, it moves to Step 6. This ensures *Done* signal is TRUE at least for 1 PLC scan cycle.

```
5: (*  Done state   *)
      Done := TRUE;
      Busy := FALSE;
      Error := FALSE;
      ErrorID:=0;
      ErrorDescrip:=' ';

      IF NOT Execute THEN
         FB_status:= 6;          //Returns to idle
      END_IF;
```

Figure 5.3.1.10 – FB State Diagram Control step 5 common for all FBs (source: own).

In Step 6, all boolean signals are set to FALSE, and *ErrorDescription* output is emptied.

```
6: (*     Done deactive if not Execute      *)
      Done := FALSE;
      Busy := FALSE;
      Error := FALSE;
      ErrorID:=0;
      ErrorDescrip:=' ';

END_CASE ;
```

Figure 5.3.1.11 – FB State Diagram Control step 6 common for all FBs (source: own).

Whether the State Diagram Control is in Step 2, therefore in *Busy* status, the Algorithm Sequence inside the Function Block is executed:

The Function Block remains in step 0 when it is idle or not in usage.

```
(* FB Algorithm                                                    *)
(* ---------------------------------------------------------------- *)

IF Busy THEN

   CASE FB_step OF
   0: (*     Idle     *)
         FB_step:=0;
```

Figure 5.3.1.12 – CBT_Connect FB algorithm step 0 when idle (source: own).

Once *Execute* input is TRUE, then FB_step is 10 and all variables are initialized.

```
10: (*   Initialization    *)
    //InternalStep variables
    TCP_Connect_Exe      :=FALSE;
    TCP_Clear_Buffer_Exe :=FALSE;
    Get_TCP_Status_Exe   :=FALSE;
    TCP_Send_Exe         :=FALSE;
    TCP_Rcv_Exe          :=FALSE;
    TCP_Close_Exe        :=FALSE;
    TON_TimeOut_Exe      :=FALSE;
    ToErrorID            :=0;
    ToErrorDescrip       :=' ';

    //FB Outputs
    CBT.WaitingReturn:=FALSE;
    CBT.Connected:=FALSE;
    Connected:=CBT.Connected;
    LastConnTime := SecToDt(0);
    LastLostTime  := SecToDt(0);
    FirstConnection:=FALSE;

    FB_step:=15;
```

Figure 5.3.1.13 – CBT_Connect FB algorithm step 10 for variables initialization (source: own).

Step 15 checks if the timeout value is in the acceptable range (between 10 and 300 milliseconds).

```
15:(*   Check Timeout value     *)

    IF (ConnectionTimeOut<10) OR (ConnectionTimeOut>300) THEN
       ToErrorID:=16#15;
       ToErrorDescrip:='Timeout range: 10~300 seconds';
       TCP_Connect_Exe:=FALSE;
       ToError_Sts :=TRUE;
    else
       FB_step:=20;
    END_IF;
```

Figure 5.3.1.14 – CBT_Connect FB algorithm step 15 checking acceptable range for timeout parameter (source: own).

Step 20 connects built-in EtherNet/IP on the PLC to the robot TCP port. If robot program in TMflow is not running when Execute input changes to TRUE, the FB is cyclically trying to connect with the robot until robot program in TMflow runs (robot connected) or timeout is exceeded (Error End).

```
20:(*    Request a connection    *)
    TCP_Connect_Exe := TRUE;
    TON_TimeOut_Exe:=TRUE;

    IF TCP_Connect.Done THEN
       TCP_Connect_Exe:=FALSE;
       TON_TimeOut_Exe:=FALSE;
       FB_step:=30;
    END_IF;

    IF TCP_Connect.Error THEN
       ToErrorID:=16#20;
       ToErrorDescrip:='Connection Error';
       TCP_Connect_Exe:=FALSE;
       ToError_Sts :=TRUE;
    END_IF;

    IF TimeOut THEN
       ToErrorID:=16#21;
       ToErrorDescrip:='TimeOut Error';
       TCP_Connect_Exe:=FALSE;
       TON_TimeOut_Exe:=FALSE;
       ToError_Sts :=TRUE;
    END_IF;
```

Figure 5.3.1.15 – CBT_Connect FB algorithm step 20 requesting connection (source: own).

Step 30 clears the receive buffer for the TCP socket on the built-in EtherNet/IP port on the PLC.

```
30: (*    Clear receive buffer    *)
    TCP_Clear_Buffer_Exe:=TRUE;

    IF TCP_Clear_Buffer.Done THEN
       TCP_Clear_Buffer_Exe:=FALSE;
       FB_step:=40;
    END_IF;

    IF TCP_Clear_Buffer.Error THEN
       ToErrorID:=16#30;
       ToErrorDescrip:='Clear Buffer Error';
       TCP_Clear_Buffer_Exe:=FALSE;
       ToError_Sts :=TRUE;
    END_IF;
```

Figure 5.3.1.16 – CBT_Connect FB algorithm step 30 clearing the TCP received buffer (source: own).

Step 40 gets the TCP connection status of the TCP socket.

```
208  40: (*   Request reading status      *)
209      Get_TCP_Status_Exe:=TRUE;
210
211      IF Get_TCP_Status.Done THEN
212         Get_TCP_Status_Exe:=FALSE;
213         FB_step:=50;
214      END_IF;
215
216      IF Get_TCP_Status.Error THEN
217         ToErrorID:=16#40;
218         ToErrorDescrip:='Get TCP Status Error';
219         Get_TCP_Status_Exe:=FALSE;
220         ToError_Sts := TRUE;
221      END_IF;
```

Figure 5.3.1.17 – CBT_Connect FB algorithm step 40 checking the status of the TCP socket (source: own).

Step 50 converts string type message in byte array format to be able to be sent.

- defines the string message:
  - *str_CheckListen:='$$TMSTA,2,00,*41RL';*
- finds the number of characters in a text string:
  - *TCP_Send_Size:=LEN(str_CheckListen);*
- separates a variable into bytes and stores them in a BYTE array:
  - *ToAryByte(In:=str_CheckListen, Order:=_eBYTE_ORDER#_LOW_HIGH, AryOut:=TCP_Send_Data[0]);*
- sends the str_CheckListen message to the robot.

Description of (*str_CheckListen:='$$TMSTA,2,00,*41RL';*):

- $$TMSTA → Communication package acquiring status // Sysmac syntax requires $$, the 1st is sysmac systax to recognize the 2nd $ as character.
- 2 → Indicates the length of 00 is 2 bytes.
- 00 → Indicates if cobot is in external script control mode or not (is Cobot in Listen node or not)
- 41 → Checksum
- $R → $R in Sysmac syntax is \R in ASCII (carriage)
- $L → $L in Sysmac syntax is \L in ASCII (enter)

```
50: (*   Request sending data     *)
    // Converts command to byte array

    str_CheckListen:='$$TMSTA,2,00,*41$R$L';
    TCP_Send_Size:=LEN(str_CheckListen);
    ToAryByte(In:=str_CheckListen, Order:=_eBYTE_ORDER#_LOW_HIGH, AryOut:=TCP_Send_Data[0]);


    TCP_Send_Exe :=TRUE;

    IF TCP_Send.Done THEN
       TCP_Send_Exe:=FALSE;
       FB_step:=60;
    END_IF;

    IF TCP_Send.Error THEN
       ToErrorID:=16#50;
       ToErrorDescrip:='TCP Send Error';
       TCP_Send_Exe:=FALSE;
       ToError_Sts := TRUE;
    END_IF;
```

Figure 5.3.1.18 – CBT_Connect FB algorithm step 50 sending the message to get connection with robot (source: own).

Step 60 reads the data from the receive buffer for a TCP socket on the built-in EtherNet/IP port on the PLC.

```
60: (*   Request receiving data      *)

    TCP_Rcv_TimeOut:=0;          //0: No timeouts
    TCP_Rcv_Size:=256;           //Set number of bytes to read from the receive buffer
    StringOfReceivedData:='';    //Clear the variable where Receive data array is compiled

    TCP_Rcv_Exe   :=TRUE;

    IF TCP_Rcv.Done THEN
       StringOfReceivedData:=AryToString(TCP_Rcv_Data[0],TCP_Rcv_RcvSize);
       TCP_Rcv_Exe:=FALSE;
       FB_step:=70;
    END_IF;

    IF TCP_Rcv.Error THEN
       ToErrorID:=16#60;
       ToErrorDescrip:='TCP Receive Error';
       TCP_Rcv_Exe:=FALSE;
       ToError_Sts := TRUE;
    END_IF;
```

Figure 5.3.1.19 – CBT_Connect FB algorithm step 60 receiving TCP message from robot (source: own).

Step 70 checks the data received from robot to TCP socket in the PLC. If the message includes the word "true", the robot is in Listen Node and the connection can be done (moves to step 80). Otherwise, if the message includes the word "false", the robot is not on Listen Node, connection cannot be done and error is triggered.

```
268  70: (*  Check received data      *)
269
270      IF FIND(StringOfReceivedData,'true') <> 0 THEN
271         FB_step:=80;
272      ELSIF FIND(StringOfReceivedData,'false') <> 0 THEN
273         ToErrorID:=16#70;
274         ToErrorDescrip:='Device is not a Cobot';
275         ToError_Sts := TRUE;
276      END_IF;
277
```

Figure 5.3.1.20 – CBT_Connect FB algorithm step 70 checking data received (source: own).

Step 80 gets the TCP connection status of the TCP socket.

```
278
279  80: (*  Continuously checking status    *)
280
281      Get_TCP_Status_Exe:=TRUE;
282
283      IF Get_TCP_Status.Done THEN
284         CBT.Connected:=TRUE;
285         Connected:=CBT.Connected;
286         Get_TCP_Status_Exe:=FALSE;
287         FB_step:=90;
288      END_IF;
289
290      IF Get_TCP_Status.Error THEN
291         ToErrorID:=16#80;
292         ToErrorDescrip:='Get TCP Status Error';
293         Get_TCP_Status_Exe:=FALSE;
294         ToError_Sts := TRUE;
295      END_IF;
296
```

Figure 5.3.1.21 – CBT_Connect FB algorithm step 80 gets TCP connection status (source: own).

Step 90 is cyclically checking if the connection is still alive, otherwise *CBT.Connected* changes to FALSE. If robot program stops when connection is stablished (the FB execution is in progress), robot will close the connection (Error End).

```
298  90: (*  Check status each time     *)
299
300      IF NOT(FirstConnection) THEN   //Updates output just once
301         LastConnTime:=GetTime();
302         FirstConnection:=TRUE;
303      END_IF;
304
305      Timer_1_Enable := TRUE ;
306
307      IF Timer_1_Done THEN
308         Timer_1_Enable:=FALSE;
309         FB_step:=FB_step-10;              //Continuously checking status
310      END_IF;
311
312      IF SocketStatus =_CLOSE_WAIT THEN
313         LastLostTime:=GetTime();
314         ToErrorID:=16#90;
315         ToErrorDescrip := 'Socket disconnected';
316         CBT.WaitingReturn:=FALSE;
317         CBT.Connected:=FALSE;
318         Connected:=CBT.Connected;
319         Get_TCP_Status_Exe:=FALSE;
320         ToError_Sts := TRUE;
321      END_IF;
322
```

Figure 5.3.1.22 – CBT_Connect FB algorithm step 90 continuously checking connection (source: own).

If *Execute* input changes to FALSE, then step 100 is executed.

```
324  100: (*  Request closing     *)
325
326      CBT.WaitingReturn:=FALSE;
327      CBT.Connected:=FALSE;
328      Connected:=CBT.Connected;
329      TCP_Close_Exe:=TRUE;
330      LastLostTime:=GetTime();
331
332      IF TCP_Close.Done THEN
333         TCP_Close_Exe:=FALSE;
334         FB_step:=110;
335      END_IF;
336
337      IF TCP_Close.Error THEN
338         ToErrorID:=16#100;
339         ToErrorDescrip:='TCP Close Error';
340         TCP_Close_Exe:=FALSE;
341         ToError_Sts := TRUE;
342      END_IF;
343
```

Figure 5.3.1.23 – CBT_Connect FB algorithm step 100 closing the TCP socket connection (source: own).

Step 110 finishes the execution of the Function Block.

```
345    110: (*  End Execution   *)
346       ToDone_Sts:=TRUE;
347
348    END_CASE ;
349
350 END_IF;
```

Figure 5.3.1.24 – CBT_Connect FB algorithm step 110 finishing the execution (source: own).

The last part of the script is reserved for TCP related FBs included be default in Sysmac Studio.

```
353 (* Function Bolcks                                                    *)
354 (* ------------------------------------------------------------------ *)
355
356  TCP_Connect(
357     Execute:=TCP_Connect_Exe,
358     SrcTcpPort:=Local_PortIP,
359     DstAdr:=AddressIP,
360     DstTcpPort:=Dest_PortIP,
361     //Done=>, Busy=>, Error=>, ErrorID=>,
362     Socket=>CBT.Socket);
363
364  TCP_Clear_Buffer(
365     Execute:=TCP_Clear_Buffer_Exe,
366     Socket:=CBT.Socket
367     //Done=>, Busy=>, Error=>, ErrorID=>
368     );
369
370  Get_TCP_Status(
371     Execute:=Get_TCP_Status_Exe,
372     Socket:=CBT.Socket,
373     //Done=>, Busy=>, Error=>, ErrorID=>,
374     TcpStatus=>SocketStatus
375     //DatRcvFlag=>
376     );
377
378  TCP_Send(
379     Execute:=TCP_Send_Exe,
380     Socket:=CBT.Socket,
381     SendDat:=TCP_Send_Data[0],
382     Size:=TCP_Send_Size
383     //Done=>, Busy=>, Error=>, ErrorID=>
384     );
385
386  TCP_Rcv(
387     Execute:=TCP_Rcv_Exe,
388     Socket:=CBT.Socket,
389     TimeOut:=TCP_Rcv_TimeOut,
390     Size:=TCP_Rcv_Size,
391     RcvDat:=TCP_Rcv_Data[0],
392     //Done=>, Busy=>, Error=>, ErrorID=>,
393     RcvSize=>TCP_Rcv_RcvSize);
394
395  TCP_Close(
396     Execute:=TCP_Close_Exe,
397     Socket:=CBT.Socket
398     //Done=>, Busy=>, Error=>, ErrorID=>
399     );
400
401  TON_TimeOut(In:=TON_TimeOut_Exe, PT:=NanoSecToTime(ConnectionTimeOut*1000000000), Q=>TimeOut);
402  Timer_1(In:=Timer_1_Enable, PT:=T#1000ms, Q=>Timer_1_Done);
403
```

Figure 5.3.1.25 – CBT_Connect FB internal functions instances for TCP communications (source: own).

Error list with the Error ID, Error Description and Action:

Error ID	Error Description	Action
15	Timeout range: 10~300 seconds	Set parameter in acceptable range
20	Connection Error	Use CBT_Connect FB for connection
21	TimeOut Error	Cobot is not in Listen Node
30	Clear Buffer Error	Check Ethernet connection wiring
40	Get TCP Status Error	Check Ethernet connection wiring
50	TCP Send Error	Check Ethernet connection wiring
60	TCP Receive Error	Check Ethernet connection wiring
70	Target device is not a Cobot	Cobot is not in Listen Node
80	Get TCP Status Error	Check Ethernet connection wiring
90	Socket disconnected	Check Ethernet connection wiring
100	TCP Close Error	Check Ethernet connection wiring

Table 5.3.1.4 – CBT_Connect error list description and action (source: own).

5.3.2. CBT_MoveAbsolute

This Function Block sends the command to move the robot to an absolute target position. Two different types of motion can be defined:

- PTP: Robot moves to the target point along the closest path of the joint angle space.
- Line: Tool moves to the target point in a straight line.

Among other parameters, user must set the Cobot In/out variable to identify the target robot with its IP address and communications port, the command ID to identify the command in the feedback sent by the robot, speed including units, acceleration time, if blending is required and if it is performed by percentage or by radius, and finally the robot arm configuration. This last parameter defines if the target position will be achieved with lefty or righty configuration, with above or below configuration and with flip or noflip configuration when the movement is defined as PTP (Point to point). Arm configuration is not available for Linear interpolated movement, in that case the arm configuration is kept as it was at the moment the motion started.

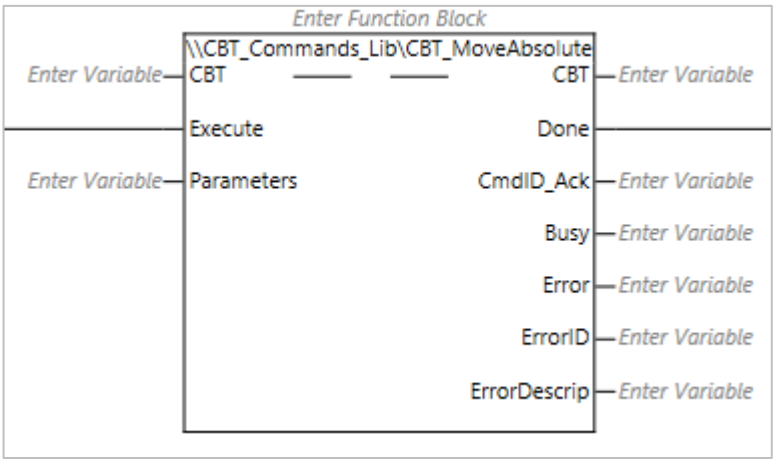

Figure 5.3.2.1 – CBT_MoveAbsolute Function Block (source: own).

In the below tables, input and output parameters are described:

Name	I/O	Data Type	Valid Range	Default	Description
CBT	In/Out	CBT_Commands_Lib\stCBT_Connection	---	---	Identifies robot IP address, port and connection.
Execute	Input	BOOL	TRUE or FALSE	FALSE	Request of instruction execution
Parameters	Input	CBT_Commands_Lib\stCBT_MovAbsParam	---	---	Motion parameters
Done	Output	BOOL	TRUE or FALSE	FALSE	TRUE when the instruction is completed
CmdID_Ack	Output	UINT	0 to +65535	0	Command identification number
Busy	Output	BOOL	TRUE or FALSE	FALSE	TRUE when the instruction is in progress
Error	Output	BOOL	TRUE or FALSE	FALSE	TRUE while there is an error
ErrorID	Output	WORD	---	16#0000	Contains the error code when an error occurs. A value of 16#0000 indicates normal execution
ErrorDescrip	Output	STRING[50]	---	---	Contains the error description when an error occurs

Table 5.3.2.1 – CBT_MoveAbsolute variables type, range, default value and description (source: own).

Name	Data Type	Valid Range	Default	Description
Socket	_sSOCKET	---	---	Socket
Connected	BOOL	TRUE or FALSE	FALSE	TRUE when robot is connected
WaitingReturn	BOOL	TRUE or FALSE	FALSE	TRUE when robot is under control

Table 5.3.2.2 – stCBT_Connection datatype, range, default value and description (source: own).

Name	Data Type	Valid Range	Default	Description
CommandID	UINT	2 to 9	0	Identification for the return message for this command
MovementCommand	CBT_Commands_Lib\ eCBT_MovCmd_Movement	---	---	Motion type
DataFormat	CBT_Commands_Lib\ eCBT_MovCmdAbs_DataFormat	---	---	Motion strategy
TargetPosition	CBT_Commands_Lib\ stCBT_Transformation	---	---	Target position
Speed	UINT	0 to 4500	0	Speed expressed as a percentage (%) or in velocity (mm/s)
AccelTime	UINT	150 to 9999	0	Time interval to accelerate to top speed (ms)
BlendingEnable	BOOL	TRUE or FALSE	FALSE	Enables blending with next motion command
BlendingValue	UINT	0 to 100	0	Blending value expressed as a percentage (%) or in radius (mm)
PrecisePositioning	BOOL	TRUE or FALSE	FALSE	Whether robot moves to the point precisely
RobotPoseEnable	BOOL	TRUE or FALSE	FALSE	Enables if arm configuration can be decided
RobotPose	CBT_Commands_Lib\ stCBT_RobotPose	---	---	Arm configuration type
ExitNode	BOOL	TRUE or FALSE	FALSE	Quit Listen node in robot program

Table 5.3.2.3 – stCBT_MovAbsParam datatype, range, default value and description (source: own).

Name	Enum Value	Description
Line	0	Motion in Joint, speed in %, blending in %
PTP	1	Motion in Cartesian, speed in %, blending in %

Table 5.3.2.4 – eCBT_MovCmd_Movement datatype, value and description (source: own).

Name	Enum Value	Description
JPP_Abs	0	Motion in Joint, speed in %, blending in %
CPP_Abs	1	Motion in Cartesian, speed in %, blending in %
CPR_Abs	2	Motion in Cartesian, speed in %, blending in radius
CAP_Abs	3	Motion in Cartesian, speed in mm/s, blending in %
CAR_Abs	4	Motion in Cartesian, speed in mm/s, blending in radius

Table 5.3.2.5 – eCBT_MovCmdAbs_DataFormat datatype, value and description (source: own).

Name	Data Type	Valid Range	Default	Description
X	REAL	---	0	X coordinates in mm
Y	REAL	---	0	Y coordinates in mm
Z	REAL	---	0	Z coordinates in mm
RX	REAL	---	0	RX coordinates in degrees
RY	REAL	---	0	RY coordinates in degrees
RZ	REAL	---	0	RZ coordinates in degrees

Table 5.3.2.6 – stCBT_Transformation datatype, range, default value and description (source: own).

Name	Data Type	Valid Range	Default	Description
Pose1	UINT	0 to 1	0	RIGHTY = 0, LEFTY = 1
Pose2	UINT	2 to 3	0	ABOVE = 2, BELOW = 3
Pose3	UINT	4 to 5	0	NOFLIP = 4, FLIP = 5

Table 5.3.2.7 – stCBT_RobotPose datatype, range, default value and description (source: own).

Function Block Script

As all Function Blocks in this Library, once the upward signal differentiation of *Execute* input is detected and it is not in *Busy* state, the sequence initialization is started, otherwise an error is triggered. Sequence initialization has been described in *5.3.1.CBT_Connect* section.

As all Function Blocks in this Library, State Diagram Control is part of the script focused on output variable operation and timing. State Diagram Control has been described in *5.3.1.CBT_Connect* section.

Hereunder, the description of the specific script of CBT_MoveAbsolute FB:

It remains in step 0 when it is idle or not in usage.

```
119
120 (* FB Algorithm                                                    *)
121 (* ---------------------------------------------------------------- *)
122
123 IF Busy THEN        //FB State Diagram control in "Busy" state
124
125     CASE FB_step OF
126     0: (*     Idle     *)
127         FB_step:=0;
```

Figure 5.3.2.2 – CBT_MoveAbsolute FB algorithm step 0 when idle (source: own).

Once *Execute* input is TRUE, then FB_step changes to 10, all variables are initialized and checks if the robot is connected and free to be used (not occupied by other FB).

```
10: (*   Initialization    *)

        TCP_Clear_Buffer_Exe  :=FALSE;
        TCP_Send_Exe          :=FALSE;
        TCP_Rcv_Exe           :=FALSE;
        ToErrorID             :=0;
        ToErrorDescrip        :=' ';
        CmdID_Ack             :=Parameters.CommandID;

        IF CBT.Connected and NOT(CBT.WaitingReturn) THEN     //If robot is connected and "released"
           FB_step:=20;
        ELSIF NOT(CBT.Connected) THEN                        //Robot not connected
           ToError_Sts := TRUE;
           ToErrorID:=16#10;
           ToErrorDescrip:='Robot not connected';
        ELSIF (CBT.WaitingReturn) THEN                       //Robot is "occupied" by other FB
           ToError_Sts := TRUE;
           ToErrorID:=16#11;
           ToErrorDescrip:='Other FB is under execution';
        END_IF;
```

Figure 5.3.2.3 – CBT_MoveAbsolute FB algorithm step 10 for variables initialization, checking connection and robot ready (source: own).

Step 20 clears the receive buffer for the TCP socket on the built-in EtherNet/IP on the PLC.

```
20: (*   Clear receive buffer    *)
        TCP_Clear_Buffer_Exe:=TRUE;

        IF TCP_Clear_Buffer.Done THEN
           TCP_Clear_Buffer_Exe:=FALSE;
           FB_step:=25;
        END_IF;

        IF TCP_Clear_Buffer.Error THEN
           TCP_Clear_Buffer_Exe:=FALSE;
           ToError_Sts := TRUE;
           ToErrorID:=16#20;
           ToErrorDescrip:='Clear buffer error';
        END_IF;
```

Figure 5.3.2.4 – CBT_MoveAbsolute FB algorithm step 20 clearing the TCP received buffer (source: own).

Step 25 verifies if the values for motion parameters are within the acceptable range.

```
25: (*   Do not exceed the seetable range  *)

    IF iParameters.MovementCommand=\\CBT_Commands_Lib\eCBT_MovCmd_Movement#Line THEN
       IF iParameters.Speed > 4500 THEN
          ToError_Sts := TRUE;
          ToErrorID:=16#25;
          ToErrorDescrip:='Speed range in Line: 0~4500 mm/s';
       END_IF;
    END_IF;

    IF iParameters.MovementCommand=\\CBT_Commands_Lib\eCBT_MovCmd_Movement#PTP THEN
       IF iParameters.Speed > 100 THEN
          ToError_Sts := TRUE;
          ToErrorID:=16#26;
          ToErrorDescrip:='Speed range in PTP: 0~100 %';
       END_IF;
    END_IF;

    IF (iParameters.AccelTime < 150) OR (iParameters.AccelTime > 9999) THEN
       ToError_Sts := TRUE;
       ToErrorID:=16#27;
       ToErrorDescrip:='Acceleration Time range is: 150~9999 ms';
    END_IF;

    IF (iParameters.CommandID < 2) OR (iParameters.CommandID > 9) THEN
       ToError_Sts := TRUE;
       ToErrorID:=16#28;
       ToErrorDescrip:='Command ID range is: 2~9';
    END_IF;

    IF iParameters.MovementCommand=\\CBT_Commands_Lib\eCBT_MovCmd_Movement#Line THEN
       IF iParameters.RobotPoseEnable = true THEN
          ToError_Sts := TRUE;
          ToErrorID:=16#29;
          ToErrorDescrip:='Robot Pose must be disable for Line command';
       END_IF;
    END_IF;

    FB_step:=30;
```

Figure 5.3.2.5 – CBT_MoveAbsolute FB algorithm step 25 checking acceptable value range for motion parameters (source: own).

Step 30 detects if the values in the arm configuration parameter are within the acceptable range.

```
209
210  30: (*   Do not exceed the seetable range  *)
211
212      IF iParameters.RobotPoseEnable THEN
213        IF (iParameters.RobotPose.Pose1 < 0) OR (iParameters.RobotPose.Pose1 > 1) THEN
214          ToError_Sts := TRUE;
215          ToErrorID:=16#30;
216          ToErrorDescrip:='Pose1 range is 0~1';
217        END_IF;
218      END_IF;
219
220      IF iParameters.RobotPoseEnable THEN
221        IF (iParameters.RobotPose.Pose2 < 2) OR (iParameters.RobotPose.Pose2 > 3) THEN
222          ToError_Sts := TRUE;
223          ToErrorID:=16#31;
224          ToErrorDescrip:='Pose2 range is 2~3';
225        END_IF;
226      END_IF;
227
228      IF iParameters.RobotPoseEnable THEN
229        IF (iParameters.RobotPose.Pose3 < 4) OR (iParameters.RobotPose.Pose3 > 5) THEN
230          ToError_Sts := TRUE;
231          ToErrorID:=16#32;
232          ToErrorDescrip:='Pose3 range is 4~5';
233        END_IF;
234      END_IF;
235
236      FB_step:=35;
```

Figure 5.3.2.6 – CBT_MoveAbsolute FB algorithm step 30 checking acceptable value range in arm configuration parameter (source: own).

Step 35 creates the corresponding string variable (str_MovementCommand) depending on the movement command parameter and the string variable (str_DataFormat) for the data format parameter.

```
35: (*  iParameters conversion to string *)

    str_CommandID:=UINT_TO_STRING(iParameters.CommandID);

    // iParameters.MovementCommand
    IF iParameters.MovementCommand=\\CBT_Commands_Lib\eCBT_MovCmd_Movement#Line THEN
       str_MovementCommand:='Line';
    END_IF;

    IF iParameters.MovementCommand=\\CBT_Commands_Lib\eCBT_MovCmd_Movement#PTP THEN
       str_MovementCommand:='PTP';
    END_IF;

    //iParameters.DataFormat
    IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdAbs_DataFormat#CAP_Abs THEN
       str_DataFormat:='CAP';
    END_IF;

    IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdAbs_DataFormat#CAR_Abs THEN
       str_DataFormat:='CAR';
    END_IF;

    IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdAbs_DataFormat#CPP_Abs THEN
       str_DataFormat:='CPP';
    END_IF;

    IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdAbs_DataFormat#CPR_Abs THEN
       str_DataFormat:='CPR';
    END_IF;

    IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdAbs_DataFormat#JPP_Abs THEN
       str_DataFormat:='JPP';
    END_IF;
```

Figure 5.3.2.7 – CBT_MoveAbsolute FB algorithm step 35 creating the string of the data format parameter (source: own).

It converts target position parameter from real to string (str_TargetPosition), speed parameter to string variable (str_Speed), acceleration time to string (str_AccelTime), blending value to string (str_BlendingValue) and precise positioning to string (str_PrecisePositioning).

```
271
272         //Conversion of REAL variables to a text string with the specified format
273         str_TP_X := RealToFormatString(In:=iParameters.TargetPosition.X, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
274         str_TP_Y := RealToFormatString(In:=iParameters.TargetPosition.Y, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
275         str_TP_Z := RealToFormatString(In:=iParameters.TargetPosition.Z, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
276         str_TP_RX := RealToFormatString(In:=iParameters.TargetPosition.RX, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
277         str_TP_RY := RealToFormatString(In:=iParameters.TargetPosition.RY, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
278         str_TP_RZ := RealToFormatString(In:=iParameters.TargetPosition.RZ, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
279         str_TP1:=CONCAT(str_TP_X,',',str_TP_Y,',',str_TP_Z);
280         str_TP2:=CONCAT(str_TP_RX,',',str_TP_RY,',',str_TP_RZ);
281         str_TargetPosition:=CONCAT(str_TP1,',',str_TP2);
282
283         //Conversion of integer to text string
284         str_Speed:=UINT_TO_STRING(iParameters.Speed);
285         str_AccelTime:=UINT_TO_STRING(iParameters.AccelTime);
286
287         IF iParameters.BlendingEnable THEN
288             str_BlendingValue:=UINT_TO_STRING(iParameters.BlendingValue);
289         ELSE
290             str_BlendingValue:='0';
291         END_IF;
292
293         //Conversion of bool to text string
294         IF iParameters.PrecisePositioning THEN
295             str_PrecisePositioning:='true';
296         ELSE
297             str_PrecisePositioning:='false';
298         END_IF;
```

Figure 5.3.2.8 – CBT_MoveAbsolute FB algorithm step 35 creating the string variables for the target position, speed value, acceleration time value, blending value and precise positioning value (source: own).

It joins the robot pose parameters (str_RobotPose) and verifies if the data format selected is acceptable for the selected movement command.

```
299
300         //iParameters.RobotPose
301         str_RobotPose:=CONCAT(  UINT_TO_STRING(iParameters.RobotPose.Pose1),',',
302                                 UINT_TO_STRING(iParameters.RobotPose.Pose2),',',
303                                 UINT_TO_STRING(iParameters.RobotPose.Pose3));
304
305
306         //Verify if dataformat input is valid for MovementCommand selection
307         IF (str_MovementCommand='PTP') THEN
308            IF (str_DataFormat='JPP') OR (str_DataFormat='CPP') THEN
309               FB_step:=70;
310            ELSE
311               ToError_Sts := TRUE;
312               ToErrorID:=16#35;
313               ToErrorDescrip:='DataFormat invalid for PTP command';
314            END_IF;
315         END_IF;
316
317         IF (str_MovementCommand='Line') THEN
318            IF (str_DataFormat='CPP') OR
319               (str_DataFormat='CPR') OR
320               (str_DataFormat='CAP') OR
321               (str_DataFormat='CAR') THEN
322               FB_step:=70;
323            ELSE
324               ToError_Sts := TRUE;
325               ToErrorID:=16#36;
326               ToErrorDescrip:='DataFormat invalid for Line command';
327            END_IF;
328         END_IF;
329
330      FB_step:=40;
```

Figure 5.3.2.9 – CBT_MoveAbsolute FB algorithm step 35 creating the string of the arm configuration parameter and checks if there is some incorrect data format parameter depending on the movement command (source: own).

Step 40 builds the message (script_command) to be sent to the robot including:

- Movement command
- Data format
- Target position
- Speed
- Acceleration time
- Blending value
- Precise positioning
- Robot pose

- If blending is disabled, QueueTag() command is included in the message. It identifies the robot motion with the CommandID number for the acknowledge when robot motion finishes the current robot motion in process. Hence, *Done* output changes to TRUE when the robot movement has been completed.
- If ExitNode is enabled, ScriptExit() command is included in the message. It exits external script control mode once the last motion command has been executed and finished. Listen Node is quitted, and not further external script functions commands are accepted.

```
40: (*  Frame building  for CheckSum calculation *)

    Header:='TMSCT';      //Header required by robot controller to receive external scripts

    //Creates the script command with all the input parameters
    IF NOT(iParameters.RobotPoseEnable) then
       Script_Command:=CONCAT(  CONCAT(str_MovementCommand,'(',str_DataFormat,','),
                                CONCAT(str_TargetPosition,',',str_Speed,','),
                                CONCAT(str_AccelTime,',',str_BlendingValue,','),
                                CONCAT(str_PrecisePositioning,')') );
    ELSE
       Script_Command:=CONCAT(  CONCAT(str_MovementCommand,'(',str_DataFormat,','),
                                CONCAT(str_TargetPosition,',',str_Speed,','),
                                CONCAT(str_AccelTime,',',str_BlendingValue,','),
                                CONCAT(str_PrecisePositioning,',',str_RobotPose,')') );
    END_IF;


    //When no blending motion, acknowledgement with QueueTag()
    IF NOT(iParameters.BlendingEnable) THEN
       Script_Command:=CONCAT(Script_Command,'$R$LQueueTag(',str_CommandID,',0)');
    END_IF;

    //Exits Listen Node in TMflow with ScriptExit()
    IF iParameters.ExitNode THEN
       Script_Command:=CONCAT(Script_Command,'$R$L','ScriptExit()');
    END_IF;
```

Figure 5.3.2.10 – CBT_MoveAbsolute FB algorithm step 40 building the message with the complete command (source: own).

Once the message has been built, it is needed to find the number of characters of (str_CommandID + Script_Command) and to store it in the string (str_Length). Then, joining all parts (Header + str_Length + str_CommandID + Script_Command) to calculate the checksum (str_Checksum_Calc).

```
360         // CheckSum calculation
361         str_Length:=INT_TO_STRING(LEN(CONCAT(str_CommandID,',',Script_Command)));
362         str_Checksum_Calc := CONCAT(CONCAT(Header,',',str_Length,',',str_CommandID),CONCAT(',',Script_Command),',');
363
364         Checksum_Length:=ToAryByte(str_Checksum_Calc,_eBYTE_ORDER#_LOW_HIGH,Send_Checksum[0]);
365
366         IF Checksum_Length>0 THEN
367            FB_step:=50;
368         ELSE
369            ToError_Sts := TRUE;
370            ToErrorID:=16#40;
371            ToErrorDescrip:='Checksum length not valid';
372         END_IF;
```

Figure 5.3.2.11 – CBT_MoveAbsolute FB algorithm step 40 building the message with the complete command and finds the number of characters in the string (source: own).

Step 50 calculates, with XOR operation, the string for the checksum (str_Checksum) and creates the complete message to be sent (str_SendFrame) composed by:

- Header
- Length of the message
- Command ID
- String command
- CheckSum calculation

```
376   50: (*  Frame building  to send command to TMflow *)
377
378         //CheckSum calculation by XOR operation
379         FOR i:=1 TO LEN(str_Checksum_Calc) BY 1 DO
380            Send_Checksum[0]:=Send_Checksum[0] XOR Send_Checksum[i];
381         END_FOR;
382
383         //Converts checksum byte value to text string
384         str_Checksum:=BYTE_TO_STRING(Send_Checksum[0]);
385         str_SendFrame := CONCAT(CONCAT('$$',Header,',',str_Length,','),CONCAT(str_CommandID,',',Script_Command),',*',str_Checksum,'$R$L');
386
387         //Finds the number of characters in the string to be sent in the frame
388         Length:=LEN(str_SendFrame);
389         Long:=ToAryByte(str_SendFrame, _eBYTE_ORDER#_LOW_HIGH,TCP_Send_Data[0]);
390
391         IF Long>0 THEN
392            FB_step:=60;
393         ELSE
394            ToError_Sts := TRUE;
395            ToErrorID:=16#50;
396            ToErrorDescrip:='Final frame building is error end';
397         END_IF;
```

Figure 5.3.2.12 – CBT_MoveAbsolute FB algorithm step 50 building the complete message (source: own).

Step 60 creates an array of strings and sends the complete message to the robot.

```
399
400  60: (*  Send command     *)
401
402      //Finds the number of characters in the string to be sent in the frame
403      TCP_Send_Size:=LEN(str_SendFrame);
404      ToAryByte(In:=str_SendFrame, Order:=_eBYTE_ORDER#_LOW_HIGH, AryOut:=TCP_Send_Data[0]);
405
406      TCP_Send_Exe :=TRUE;
407
408      IF TCP_Send.Done THEN
409         CBT.WaitingReturn:=TRUE;    //Flag to set the robot in "busy" state to avoid other FB to be executed
410         TCP_Send_Exe:=FALSE;
411         FB_step:=70;
412      END_IF;
413
414      IF TCP_Send.Error THEN
415         TCP_Send_Exe:=FALSE;
416         ToError_Sts := TRUE;
417         ToErrorID:=16#60;
418         ToErrorDescrip:='TCP send error';
419      END_IF;
```

Figure 5.3.2.13 – CBT_MoveAbsolute FB algorithm step 60 sending the complete message (source: own).

Step 70 reads the data from the receive buffer for the TCP socket on the built-in EtherNet/IP on the PLC.

```
421
422  70: (*  Request receiving data     *)
423
424      TCP_Rcv_TimeOut:=0;          //0: No timeouts
425      TCP_Rcv_Size:=256;           //Set number of bytes to read from the receive buffer
426      StringOfReceivedData:='';    //Clear the variable where Receive data array is compiled
427
428      TCP_Rcv_Exe :=TRUE;
429
430      IF TCP_Rcv.Done THEN
431         StringOfReceivedData:=AryToString(TCP_Rcv_Data[0],TCP_Rcv_Size);
432         TCP_Rcv_Exe:=FALSE;
433         FB_step:=80;
434      END_IF;
435
436      IF TCP_Rcv.Error THEN
437         TCP_Rcv_Exe:=FALSE;
438         ToError_Sts := TRUE;
439         ToErrorID:=16#70;
440         ToErrorDescrip:='TCP receive error';
441      END_IF;
```

Figure 5.3.2.14 – CBT_MoveAbsolute FB algorithm step 70 receiving TCP message from robot (source: own).

Step 80 checks the data received in the TCP socket. If the message includes the word "OK", the command has been accepted by the robot.

If blending is disabled, *Done* output will remain FALSE until the motion command has been completed and acknowledged by the robot. FB sequence then jumps to step 90.

If blending is enabled, *Done* output is TRUE at this moment (does not wait until motion completed). FB sequence then jumps to step 200.

```
80: (*   Check acknowledgement Command accepted     *)

   IF FIND(StringOfReceivedData,'TMSTA') <> 0 THEN        //Message no vaild
      FB_step:=70;
   END_IF;

   IF FIND(StringOfReceivedData,'TMSCT') <> 0 THEN
      IF FIND(StringOfReceivedData,'OK') <> 0 THEN        //Command accepted
         IF NOT(iParameters.BlendingEnable) THEN
            FB_step:=90;                                  //"Done" signal waits until motion is finished (no blending)
         ELSE
            CBT.WaitingReturn:=FALSE;                     //Flag to set the robot in "released" state to avoid other FB to be executed
            CmdID_Ack:=STRING_TO_UINT(str_CommandID); //Output the Command ID when ack is done
            FB_step:=200;                                 //"Done" signal does not wait until  motion ends (allows blending)
         END_IF;
      ELSIF FIND(StringOfReceivedData,'ERROR') <> 0 THEN
         CBT.WaitingReturn:=FALSE;                        //Flag to set the robot in "released" state to avoid other FB to be executed
         ToError_Sts := TRUE;
         ToErrorID:=16#80;
         ToErrorDescrip:=' Command rejected';
      END_IF;
   END_IF;
```

Figure 5.3.2.15 – CBT_MoveAbsolute FB algorithm step 80 checking what information has been received (source: own).

Step 90 clears the buffer of received data because Blending is disabled, and the FB is waiting for return message from robot as acknowledge of motion completed.

```
90: (*   Clear receive buffer    *)
      TCP_Clear_Buffer_Exe:=TRUE;

      IF TCP_Clear_Buffer.Done THEN
         TCP_Clear_Buffer_Exe:=FALSE;
         FB_step:=100;
      END_IF;

      IF TCP_Clear_Buffer.Error THEN
         ToError_Sts := TRUE;
         ToErrorID:=16#90;
         ToErrorDescrip:='Clear buffer error';
      END_IF;
```

Figure 5.3.2.16 – CBT_MoveAbsolute FB algorithm step 90 clearing the buffer of received data (source: own).

Step 100 reads the data from the receive buffer for the TCP socket on the built-in EtherNet/IP on the PLC.

```
100: (* Request receiving data      *)

    CmdID_Ack:=STRING_TO_UINT(str_CommandID);    //Output the Command ID when ack is done

    TCP_Rcv_TimeOut:=0;          //0: No timeouts
    TCP_Rcv_Size:=256;           //Set number of bytes to read from the receive buffer
    StringOfReceivedData:='';    //Clear the variable where Receive data array is compiled

    TCP_Rcv_Exe :=TRUE;

    IF TCP_Rcv.Done THEN
       StringOfReceivedData:=AryToString(TCP_Rcv_Data[0],TCP_Rcv_Size);
       TCP_Rcv_Exe:=FALSE;
       FB_step:=110;
    END_IF;

    IF TCP_Rcv.Error THEN
       TCP_Rcv_Exe:=FALSE;
       ToError_Sts := TRUE;
       ToErrorID:=16#100;
       ToErrorDescrip:='TCP receive error';
    END_IF;
```

Figure 5.3.2.17 – CBT_MoveAbsolute FB algorithm step 100 receiving TCP message from robot (source: own).

Step 110 verifies the data received in the TCP socket. If the message includes *TMSTA,* it means this message contains acknowledge information of the motion command identified with *str_CommandID*. If message contains the word “true”, the motion has been completed. Otherwise, and error occurred.

```
110: (* Check acknowledgement Motion Completed    *)

    IF FIND(StringOfReceivedData,'TMSTA') <> 0 THEN              //QueueTag acknowledgement
       IF FIND(StringOfReceivedData,str_CommandID) <> 0 THEN     //QueueTag for the last motion command
          IF FIND(StringOfReceivedData,'true') <> 0 THEN         //Motion finished
             CBT.WaitingReturn:=FALSE;                           //Flag to set the robot in "released" state to avoid other FB to be executed
             CmdID_Ack:=STRING_TO_UINT(str_CommandID);           //Output the Command ID when ack is done
             FB_step:=200;
          ELSIF FIND(StringOfReceivedData,'false') <> 0 THEN
             CBT.WaitingReturn:=FALSE;                           //Flag to set the robot in "released" state to avoid other FB to be executed
             ToError_Sts := TRUE;
             ToErrorID:=16#110;
             ToErrorDescrip:=' Motion failed';
          END_IF;
       ELSE    //no str_CommandID
          FB_step:=100;
       END_IF;
    ELSE       //no TMSTA
       FB_step:=100;
    END_IF;
```

Figure 5.3.2.18 – CBT_MoveAbsolute FB algorithm step 110 verifying data received and acknowledge (source: own).

Step 200 finishes the execution of the Function Block.

```
528    200: (*     End Execution *)
529        ToDone_Sts:=TRUE;
530
531    END_CASE ;
532
533  END_IF;
534
```

Figure 5.3.2.19 – CBT_MoveAbsolute FB algorithm step 200 finishing the execution (source: own).

The last part of the script is reserved for TCP related FBs included be default in Sysmac Studio.

```
538 (* Function Bolcks                                                    *)
539 (* ------------------------------------------------------------------ *)
540
541
542  TCP_Clear_Buffer(
543     Execute:=TCP_Clear_Buffer_Exe,
544     Socket:=CBT.Socket
545     //Done=>, Busy=>, Error=>, ErrorID=>
546     );
547
548  TCP_Send(
549     Execute:=TCP_Send_Exe,
550     Socket:=CBT.Socket,
551     SendDat:=TCP_Send_Data[0],
552     Size:=TCP_Send_Size
553     //Done=>, Busy=>, Error=>, ErrorID=>
554     );
555
556  TCP_Rcv(
557     Execute:=TCP_Rcv_Exe,
558     Socket:=CBT.Socket,
559     TimeOut:=TCP_Rcv_TimeOut,
560     Size:=TCP_Rcv_Size,
561     RcvDat:=TCP_Rcv_Data[0],
562     //Done=>, Busy=>, Error=>, ErrorID=>,
563     RcvSize=>TCP_Rcv_RcvSize);
564
```

Figure 5.3.2.20 – CBT_MoveAbsolute FB internal functions instances for TCP communications (source: own).

Error list with the Error ID, Error Description and Action:

Error ID	Error Description	Action
10	Robot not connected	Use CBT_Connect FB for connection
11	Other FB is under execution	Wait until other FB is done
20	Clear buffer error	Check Ethernet connection wiring
25	Speed range in Line: 0~4500 mm/s	Set parameter in acceptable range
26	Speed range in PTP: 0~100 %	Set parameter in acceptable range
27	Acceleration Time range is: 150~9999 ms	Set parameter in acceptable range
28	Command ID range is: 2~9	Set parameter in acceptable range
29	Robot Pose must be disabled for Line command	Set parameter in acceptable range
30	Pose1 range is 0~1	Set parameter in acceptable range
31	Pose2 range is 2~3	Set parameter in acceptable range
32	Pose3 range is 4~5	Set parameter in acceptable range
35	DataFormat invalid for PTP command	Set parameter in acceptable range
36	DataFormat invalid for Line command	Set parameter in acceptable range
40	Checksum length not valid	Re-execution is required
50	Final frame building is error end	Re-execution is required
60	TCP send error	Check Ethernet connection wiring
70	TCP receive error	Check Ethernet connection wiring
80	Command rejected	Re-execution is required
90	Clear buffer error	Check Ethernet connection wiring
100	TCP receive error	Check Ethernet connection wiring
110	Motion failed	Re-execution is required

Table 5.3.2.8 – CBT_MoveAbsolute error list description and action (source: own).

5.3.3. CBT_MoveRelative

This Function Block sends the command to move the robot to a relative target position. Two different types of motion can be defined:

- PTP: Robot moves to the target point along the closest path of the joint angle space.
- Line: Tool moves to the target point in a straight line.

Among other parameters, user must set the Cobot In/out variable to identify the target robot with its IP address and communications port, the command ID to identify the command in the feedback sent by the robot, speed including units, acceleration time, if blending is required and if it is performed by percentage or by radius.

Unlike absolute movements with CBT_MoveAbsolute, arm configuration is not available for relative movements, in this case the arm configuration is kept as it was at the moment the motion started.

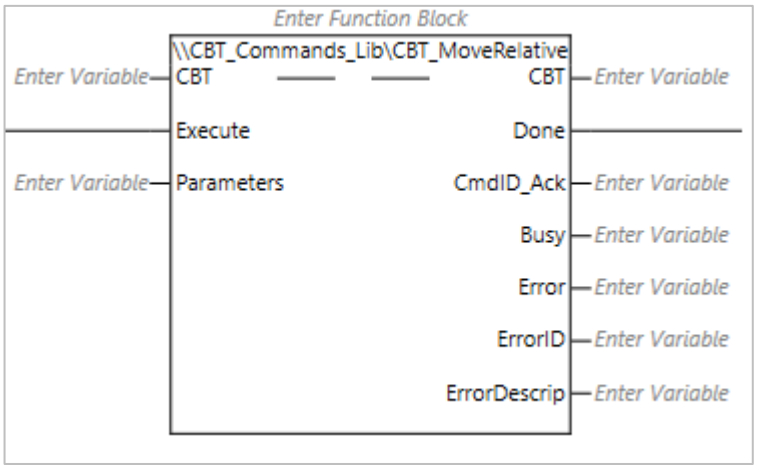

Figure 5.3.3.1 – CBT_MoveRelative Function Block (source: own).

In the below tables, input and output parameters are described:

Name	I/O	Data Type	Valid Range	Default	Description
CBT	In/Out	CBT_Commands_Lib\stCBT_Connection	---	---	Identifies robot IP address, port and connection.
Execute	Input	BOOL	TRUE or FALSE	FALSE	Request of instruction execution
Parameters	Input	CBT_Commands_Lib\stCBT_MovRelParam	---	---	Motion parameters
Done	Output	BOOL	TRUE or FALSE	FALSE	TRUE when the instruction is completed
CmdID_Ack	Output	UINT	0 to +65535	0	Command identification number
Busy	Output	BOOL	TRUE or FALSE	FALSE	TRUE when the instruction is in progress
Error	Output	BOOL	TRUE or FALSE	FALSE	TRUE while there is an error
ErrorID	Output	WORD	---	16#0000	Contains the error code when an error occurs. A value of 16#0000 indicates normal execution
ErrorDescrip	Output	STRING[50]	---	---	Contains the error description when an error occurs

Table 5.3.3.1 – CBT_MoveRelative variables type, range, default value and description (source: own).

Name	Data Type	Valid Range	Default	Description
Socket	_sSOCKET	---	---	Socket
Connected	BOOL	TRUE or FALSE	FALSE	TRUE when robot is connected
WaitingReturn	BOOL	TRUE or FALSE	FALSE	TRUE when robot is under control

Table 5.3.3.2 – stCBT_Connection datatype, range, default value and description (source: own).

Name	Data Type	Valid Range	Default	Description
CommandID	UINT	2 to 9	0	Identification for the return message for this command
MovementCommand	CBT_Commands_Lib\ eCBT_MovCmd_Movement	---	---	Motion type
DataFormat	CBT_Commands_Lib\ eCBT_MovCmdRel_DataFormat	---	---	Motion strategy
TargetPosition	CBT_Commands_Lib\ stCBT_Transformation	---	---	Target position
Speed	UINT	0 to 4500	0	Speed expressed as a percentage (%) or in velocity (mm/s)
AccelTime	UINT	150 to 9999	0	Time interval to accelerate to top speed (ms)
BlendingEnable	BOOL	TRUE or FALSE	FALSE	Enables blending with next motion command
BlendingValue	UINT	0 to 100	0	Blending value expressed as a percentage (%) or in radius (mm)
PrecisePositioning	BOOL	TRUE or FALSE	FALSE	Whether robot moves to the point precisely
ExitNode	BOOL	TRUE or FALSE	FALSE	Quit Listen node in robot program

Table 5.3.3.3 – stCBT_MovRelParam datatype, range, default value and description (source: own).

Name	Enum Value	Description
Line	0	Motion in Joint, speed in %, blending in %
PTP	1	Motion in Cartesian, speed in %, blending in %

Table 5.3.3.4 – eCBT_MovCmd_Movement datatype, value and description (source: own).

Name	Enum Value	Description
JPP_Rel	0	Motion in Joint, speed in %, blending in %
CPP_Rel	1	Motion in Cartesian, speed in %, blending in %
CPR_Rel	2	Motion in Cartesian, speed in %, blending in radius
CAP_Rel	3	Motion in Cartesian, speed in mm/s, blending in %
CAR_Rel	4	Motion in Cartesian, speed in mm/s, blending in radius
TPP_Rel	5	Motion in Tool, speed in %, blending in %
TPR_Rel	6	Motion in Tool, speed in %, blending in radius
TAP_Rel	7	Motion in Tool, speed in mm/s, blending in %
TAR_Rel	8	Motion in Tool, speed in mm/s, blending in radius

Table 5.3.3.5 – eCBT_MovCmdRel_DataFormat datatype, value and description (source: own).

Name	Data Type	Valid Range	Default	Description
X	REAL	---	0	X coordinates in mm
Y	REAL	---	0	Y coordinates in mm
Z	REAL	---	0	Z coordinates in mm
RX	REAL	---	0	RX coordinates in degrees
RY	REAL	---	0	RY coordinates in degrees
RZ	REAL	---	0	RZ coordinates in degrees

Table 5.3.3.6 – stCBT_Transformation datatype, range, default value and description (source: own).

Function Block Script

As all Function Blocks in this Library, once the upward signal differentiation of *Execute* input is detected and it is not in *Busy* state, the sequence initialization is started, otherwise an error is triggered. Sequence initialization has been described in *5.3.1.CBT_Connect* section.

As all Function Blocks in this Library, State Diagram Control is part of the script focused on output variable operation and timing. State Diagram Control has been described in *5.3.1.CBT_Connect* section.

From step 0 to 30 and from step 60 to the end, refer to the section *5.3.2. CBT_MoveAbsolute*.

Hereunder, the description of the specific script of CBT_MoveRelative FB:

Step 30 creates the corresponding string variable (str_MovementCommand) depending on the movement command parameter and the string variable (str_DataFormat) for the data format parameter.

```
30: (*  iParameters conversion to string *)

    str_CommandID:=UINT_TO_STRING(iParameters.CommandID);

    // iParameters.MovementCommand
    IF iParameters.MovementCommand=\\CBT_Commands_Lib\eCBT_MovCmd_Movement#Line THEN
       str_MovementCommand:='Move_Line';
    END_IF;

    IF iParameters.MovementCommand=\\CBT_Commands_Lib\eCBT_MovCmd_Movement#PTP THEN
       str_MovementCommand:='Move_PTP';
    END_IF;

    //iParameters.DataFormat
    IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#JPP_Rel THEN
       str_DataFormat:='JPP';
    END_IF;

    IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#CAP_Rel THEN
       str_DataFormat:='CAP';
    END_IF;

    IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#CAR_Rel THEN
       str_DataFormat:='CAR';
    END_IF;

    IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#CPP_Rel THEN
       str_DataFormat:='CPP';
    END_IF;

    IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#CPR_Rel THEN
       str_DataFormat:='CPR';
    END_IF;

    IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#TAP_Rel THEN
       str_DataFormat:='TAP';
    END_IF;

    IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#TAR_Rel THEN
       str_DataFormat:='TAR';
    END_IF;

    IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#TPP_Rel THEN
       str_DataFormat:='TPP';
    END_IF;

    IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#TPR_Rel THEN
       str_DataFormat:='TPR';
    END_IF;
```

Figure 5.3.3.2 – CBT_MoveRelative FB algorithm step 30 creating the string of the data format parameter (source: own).

It converts target position parameter from real to string (str_TargetPosition), speed parameter to string variable (str_Speed), acceleration time to string (str_AccelTime), blending value to string (str_BlendingValue) and precise positioning to string (str_PrecisePositioning).

```
250
251         str_TP_X := RealToFormatString(In:=iParameters.TargetPosition.X, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
252         str_TP_Y := RealToFormatString(In:=iParameters.TargetPosition.Y, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
253         str_TP_Z := RealToFormatString(In:=iParameters.TargetPosition.Z, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
254         str_TP_RX := RealToFormatString(In:=iParameters.TargetPosition.RX, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
255         str_TP_RY := RealToFormatString(In:=iParameters.TargetPosition.RY, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
256         str_TP_RZ := RealToFormatString(In:=iParameters.TargetPosition.RZ, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
257         str_TP1:=CONCAT(str_TP_X,',',str_TP_Y,',',str_TP_Z);
258         str_TP2:=CONCAT(str_TP_RX,',',str_TP_RY,',',str_TP_RZ);
259         str_TargetPosition:=CONCAT(str_TP1,',',str_TP2);
260
261         str_Speed:=UINT_TO_STRING(iParameters.Speed);
262         str_AccelTime:=UINT_TO_STRING(iParameters.AccelTime);
263
264         IF iParameters.BlendingEnable THEN
265             str_BlendingValue:=UINT_TO_STRING(iParameters.BlendingValue);
266         ELSE
267             str_BlendingValue:='0';
268         END_IF;
269
270         IF iParameters.PrecisePositioning THEN
271             str_PrecisePositioning:='true';
272         ELSE
273             str_PrecisePositioning:='false';
274         END_IF;
```

Figure 5.3.3.3 – CBT_MoveRelative FB algorithm step 30 creating the string of the target position, speed value, acceleration time value, blending value and precise positioning value (source: own).

Verifies if the data format selected is acceptable for the selected movement command.

```
276         IF (str_MovementCommand='Move_PTP') THEN
277             IF (str_DataFormat='JPP') OR
278                 (str_DataFormat='CPP') OR
279                 (str_DataFormat='TPP') THEN
280                 FB_step:=70;
281             ELSE
282                 ToError_Sts := TRUE;
283                 ToErrorID:=16#30;
284                 ToErrorDescrip:='DataFormat invalid for PTP command';
285             END_IF;
286         END_IF;
287
288         IF (str_MovementCommand='Move_Line') THEN
289             IF (str_DataFormat='CPP') OR
290                 (str_DataFormat='CPR') OR
291                 (str_DataFormat='CAP') OR
292                 (str_DataFormat='CAR') OR
293                 (str_DataFormat='TPP') OR
294                 (str_DataFormat='TPR') OR
295                 (str_DataFormat='TAP') OR
296                 (str_DataFormat='TAR') THEN
297                 FB_step:=70;
298             ELSE
299                 ToError_Sts := TRUE;
300                 ToErrorID:=16#31;
301                 ToErrorDescrip:='DataFormat invalid for Line command';
302             END_IF;
303         END_IF;
304
305       FB_step:=40;
```

Figure 5.3.3.4 – CBT_MoveRelative FB algorithm step 30 checking if there is some incorrect data format parameter depending on the movement command (source: own).

Step 40 builds the message (script_command) to be sent to the robot including:

- Movement command
- Data format
- Target position
- Speed
- Acceleration time
- Blending value
- Precise positioning

- If blending is disabled, QueueTag() command is included in the message. It identifies the robot motion with the CommandID number for the acknowledge when robot motion finishes the current robot motion in process. Hence, *Done* output changes to TRUE when the robot movement has been completed.
- If ExitNode is enabled, ScriptExit() command is included in the message. It exits external script control mode once the last motion command has been executed and finished. Listen Node is quitted, and not further external script functions commands are accepted.

```
40: (*  Frame building  for CheckSum calculation *)

    Header:='TMSCT';

    Script_Command:=CONCAT(  CONCAT(str_MovementCommand,'(',str_DataFormat,','),
                             CONCAT(str_TargetPosition,',',str_Speed,','),
                             CONCAT(str_AccelTime,',',str_BlendingValue,','),
                             CONCAT(str_PrecisePositioning,')') );

    //When no blending Motion ended acknowledgement with QueueTag()
    IF NOT(iParameters.BlendingEnable) THEN
       Script_Command:=CONCAT(Script_Command,'$R$LQueueTag(',str_CommandID,',0)');
    END_IF;

    //Exits Listen Node in TMflow with ScriptExit()
    IF iParameters.ExitNode THEN
       Script_Command:=CONCAT(Script_Command,'$R$L','ScriptExit()');
    END_IF;
```

Figure 5.3.3.5 – CBT_MoveRelative FB algorithm step 40 building the message with the complete command (source: own).

Once the message has been built, it is needed to find the number of characters of (str_CommandID + Script_Command) and to store it in the string (str_Length). Then, joining all parts (Header + str_Length + str_CommandID + Script_Command) to calculate the checksum (str_Checksum_Calc).

```
325
326         // CheckSum calculation
327         str_Length:=INT_TO_STRING(LEN(CONCAT(str_CommandID,',',Script_Command)));
328         str_Checksum_Calc := CONCAT(CONCAT(Header,',',str_Length,',',str_CommandID),CONCAT(',',Script_Command),',');
329
330         Checksum_Length:=ToAryByte(str_Checksum_Calc, _eBYTE_ORDER#_LOW_HIGH,Send_Checksum[0]);
331
332         IF Checksum_Length>0 THEN
333            FB_step:=50;
334         ELSE
335            ToError_Sts := TRUE;
336            ToErrorID:=16#40;
337            ToErrorDescrip:='Checksum length not valid';
338         END_IF;
339
```

Figure 5.3.3.6 – CBT_MoveRelative FB algorithm step 40 building the message with the complete command and finds the number of characters in the string (source: own).

Step 50 calculates, with XOR operation, the string for the checksum (str_Checksum) and created the complete message to be sent (str_SendFrame) composed by:

- Header
- Length of the message
- Command ID
- String command
- CheckSum calculation

```
341
342 50: (*  Frame building  to send command to TMflow *)
343
344         FOR i:=1 TO LEN(str_Checksum_Calc) BY 1 DO
345            Send_Checksum[0]:=Send_Checksum[0] XOR Send_Checksum[i];
346         END_FOR;
347
348
349         str_Checksum:=BYTE_TO_STRING(Send_Checksum[0]);
350         str_SendFrame := CONCAT(CONCAT('$$',Header,',',str_Length,','),CONCAT(str_CommandID,',',Script_Command),',*',str_Checksum,'$R$L');
351
352         Length:=LEN(str_SendFrame);
353         Long:=ToAryByte(str_SendFrame, _eBYTE_ORDER#_LOW_HIGH,TCP_Send_Data[0]);
354
355         IF Long>0 THEN
356            FB_step:=60;
357         ELSE
358            ToError_Sts := TRUE;
359            ToErrorID:=16#50;
360            ToErrorDescrip:='Final frame building is error end';
361         END_IF;
362
```

Figure 5.3.3.7 – CBT_MoveRelative FB algorithm step 50 builds the complete message (source: own).

From step 60 to the end of the algorithm is already described in previous FB explanation.

Error list with the Error ID, Error Description and Action:

Error ID	Error Description	Action
10	Robot not connected	Use CBT_Connect FB for connection
11	Other FB is under execution	Wait until other FB is done
20	Clear buffer error	Check Ethernet connection wiring
25	Speed range in Line: 0~4500 mm/s	Set parameter in acceptable range
26	Speed range in PTP: 0~100 %	Set parameter in acceptable range
27	Acceleration Time range is: 150~9999 ms	Set parameter in acceptable range
28	Command ID range is: 2~9	Set parameter in acceptable range
30	DataFormat invalid for PTP command	Set parameter in acceptable range
31	DataFormat invalid for Line command	Set parameter in acceptable range
40	Checksum length not valid	Re-execution is required
50	Final frame building is error end	Re-execution is required
60	TCP send error	Check Ethernet connection wiring
70	TCP receive error	Check Ethernet connection wiring
80	Command rejected	Re-execution is required
90	Clear buffer error	Check Ethernet connection wiring
100	TCP receive error	Check Ethernet connection wiring
110	Motion failed	Re-execution is required

Table 5.3.3.7 – CBT_MoveRelative error list description and action (source: own).

5.3.4. CBT_MoveCircle

This Function Block sends the command to move the robot describing a circular movement defined by the current position as initial position, an intermediate position, and an end position. The arc length must be also determined.

Among other parameters, user must set the Cobot In/out variable to identify the target robot with its IP address and communications port, the command ID to identify the command in the feedback sent by the robot, speed including units, acceleration time, if blending is required and if it is performed by percentage or by radius.

Unlike absolute movements with CBT_MoveAbsolute, arm configuration is not available for circular movements because, like in linear interpolation movements, the position of the TCP is constantly monitored, and it must follow the precalculated path. Hence, the arm configuration is kept as it was at the moment the motion started.

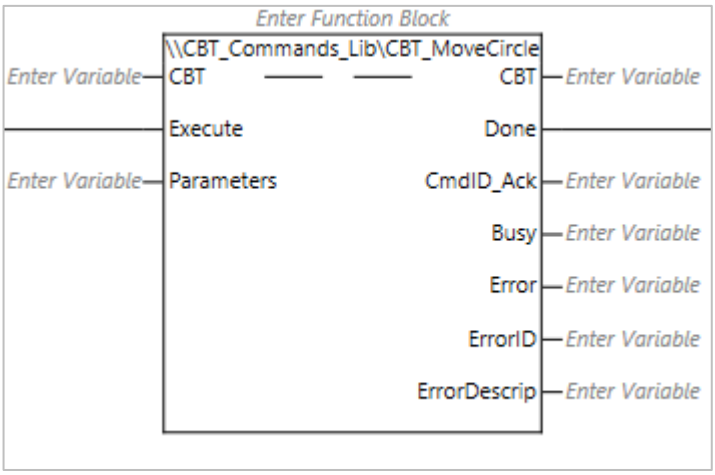

Figure 5.3.4.1 – CBT_MoveCircle Function Block (source: own).

In the below tables, input and output parameters are described:

Name	I/O	Data Type	Valid Range	Default	Description
CBT	In/Out	CBT_Commands_Lib\stCBT_Connection	---	---	Identifies robot IP address, port and connection.
Execute	Input	BOOL	TRUE or FALSE	FALSE	Request of instruction execution
Parameters	Input	CBT_Commands_Lib\stCBT_MovCircleParam	---	---	Motion parameters
Done	Output	BOOL	TRUE or FALSE	FALSE	TRUE when the instruction is completed
CmdID_Ack	Output	UINT	0 to +65535	0	Command identification number
Busy	Output	BOOL	TRUE or FALSE	FALSE	TRUE when the instruction is in progress
Error	Output	BOOL	TRUE or FALSE	FALSE	TRUE while there is an error
ErrorID	Output	WORD	---	16#0000	Contains the error code when an error occurs. A value of 16#0000 indicates normal execution
ErrorDescrip	Output	STRING[50]	---	---	Contains the error description when an error occurs

Table 5.3.4.1 – CBT_MoveCircle variables, range, default value and description (source: own).

Name	Data Type	Valid Range	Default	Description
Socket	_sSOCKET	---	---	Socket
Connected	BOOL	TRUE or FALSE	FALSE	TRUE when robot is connected
WaitingReturn	BOOL	TRUE or FALSE	FALSE	TRUE when robot is under control

Table 5.3.4.2 – stCBT_Connection datatype, range, default value and description (source: own).

Name	Data Type	Valid Range	Default	Description
CommandID	UINT	2 to 9	0	Identification for the return message for this command
DataFormat	CBT_Commands_Lib\ eCBT_MovCmdCircle_DataFormat	---	---	Motion strategy
ArcPoint	CBT_Commands_Lib\ stCBT_Transformation	---	---	A point on arc
EndPoint	CBT_Commands_Lib\ stCBT_Transformation	---	---	The end point of arc
Speed	UINT	0 to 4500	0	Speed expressed as a percentage (%) or in velocity (mm/s)
AccelTime	UINT	150 to 9999	0	Time interval to accelerate to top speed (ms)
BlendingEnable	BOOL	TRUE or FALSE	FALSE	Enables blending with next motion command
BlendingValue	UINT	0 to 100	0	Blending value expressed as a percentage (%) or in radius (mm)
ArcAngle	UINT	0 to 360	0	Length of the arc
PrecisePositioning	BOOL	TRUE or FALSE	FALSE	Whether robot moves to the point precisely
ExitNode	BOOL	TRUE or FALSE	FALSE	Quit Listen node in robot program

Table 5.3.4.3 – stCBT_MovCircleParam datatype, range, default value and description (source: own).

Name	Enum Value	Description
CPP_Circle	0	Motion in Cartesian, speed in %, blending in %
CAP_Circle	1	Motion in Cartesian, speed in mm/s, blending in %

Table 5.3.4.4 – eCBT_MovCmdCircle_DataFormat datatype, value and description (source: own).

Name	Data Type	Valid Range	Default	Description
X	REAL	---	0	X coordinates in mm
Y	REAL	---	0	Y coordinates in mm
Z	REAL	---	0	Z coordinates in mm
RX	REAL	---	0	RX coordinates in degrees
RY	REAL	---	0	RY coordinates in degrees
RZ	REAL	---	0	RZ coordinates in degrees

Table 5.3.4.5 – stCBT_Transformation datatype, range, default value and description (source: own).

Function Block Script

As all Function Blocks in this Library, once the upward signal differentiation of *Execute* input is detected and it is not in *Busy* state, the sequence initialization is started, otherwise an error is triggered. Sequence initialization has been described in *5.3.1.CBT_Connect* section.

As all Function Blocks in this Library, State Diagram Control is part of the script focused on output variable operation and timing. State Diagram Control has been described in *5.3.1.CBT_Connect* section.

From step 0 to 30 and from step 60 to the end, refer to the section *5.3.2. CBT_MoveAbsolute*.

Hereunder, the description of the specific script of CBT_MoveCircle FB:

Step 30 creates the corresponding string variable (str_MovementCommand) and the string variable (str_DataFormat) for the data format parameter.

```
199  30: (*  iParameters conversion to string *)
200
201      str_CommandID:=UINT_TO_STRING(iParameters.CommandID);
202
203      str_MovementCommand:='Circle';
204
205      //iParameters.DataFormat
206      IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdCircle_DataFormat#CAP_Circle THEN
207         str_DataFormat:='CAP';
208      END_IF;
209
210      IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdCircle_DataFormat#CPP_Circle THEN
211         str_DataFormat:='CPP';
212      END_IF;
```

Figure 5.3.4.2 – CBT_MoveCircle FB algorithm step 30 creating the string of the data format parameter (source: own).

It converts arc position and end point parameters from real to string (str_ArcPosition and str_EndPoint), speed parameter to string variable (str_Speed), acceleration time to string (str_AccelTime), blending value to string (str_BlendingValue) and precise positioning to string (str_PrecisePositioning).

```
214         str_AP_X := RealToFormatString(In:=iParameters.ArcPoint.X, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
215         str_AP_Y := RealToFormatString(In:=iParameters.ArcPoint.Y, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
216         str_AP_Z := RealToFormatString(In:=iParameters.ArcPoint.Z, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
217         str_AP_RX := RealToFormatString(In:=iParameters.ArcPoint.RX, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
218         str_AP_RY := RealToFormatString(In:=iParameters.ArcPoint.RY, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
219         str_AP_RZ := RealToFormatString(In:=iParameters.ArcPoint.RZ, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
220         str_AP1:=CONCAT(str_AP_X,',',str_AP_Y,',',str_AP_Z);
221         str_AP2:=CONCAT(str_AP_RX,',',str_AP_RY,',',str_AP_RZ);
222         str_ArcPoint:=CONCAT(str_AP1,',',str_AP2);
223
224         str_EP_X := RealToFormatString(In:=iParameters.EndPoint.X, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
225         str_EP_Y := RealToFormatString(In:=iParameters.EndPoint.Y, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
226         str_EP_Z := RealToFormatString(In:=iParameters.EndPoint.Z, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
227         str_EP_RX := RealToFormatString(In:=iParameters.EndPoint.RX, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
228         str_EP_RY := RealToFormatString(In:=iParameters.EndPoint.RY, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
229         str_EP_RZ := RealToFormatString(In:=iParameters.EndPoint.RZ, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
230         str_EP1:=CONCAT(str_EP_X,',',str_EP_Y,',',str_EP_Z);
231         str_EP2:=CONCAT(str_EP_RX,',',str_EP_RY,',',str_EP_RZ);
232         str_EndPoint:=CONCAT(str_EP1,',',str_EP2);
233
234         str_Speed:=UINT_TO_STRING(iParameters.Speed);
235         str_AccelTime:=UINT_TO_STRING(iParameters.AccelTime);
236
237         IF iParameters.BlendingEnable THEN
238             str_BlendingValue:=UINT_TO_STRING(iParameters.BlendingValue);
239         ELSE
240             str_BlendingValue:='0';
241         END_IF;
242
243         str_AcrAngle:=UINT_TO_STRING(iParameters.ArcAngle);
244
245         IF iParameters.PrecisePositioning THEN
246             str_PrecisePositioning:='true';
247         ELSE
248             str_PrecisePositioning:='false';
249         END_IF;
250
251     FB_step:=40;
```

Figure 5.3.4.3 – CBT_MoveCircle FB algorithm step 30 creating the string of the target position, speed value, acceleration time value, blending value and precise positioning value (source: own).

Step 40 builds the message (script_command) to be sent to the robot including:

- Movement command
- Data format
- Arc point
- End point
- Speed
- Acceleration time
- Blending value
- Precise positioning

- If blending is disabled, QueueTag() command is included in the message. It identifies the robot motion with the CommandID number for the acknowlegde when robot motion finishes the current robot motion in process. Hence, *Done* output changes to TRUE when the robot movement has been completed.
- If ExitNode is enabled, ScriptExit() command is included in the message. It exits external script control mode once the last motion command has been executed and finished. Listen Node is quitted, and not further external script functions commands are accepted.

```
253  40: (*  Frame building  for CheckSum calculation *)
254
255       Header:='TMSCT';
256
257       Script_Command:=CONCAT(  CONCAT(str_MovementCommand,'(',str_DataFormat,','),
258                                CONCAT(str_ArcPoint,','),
259                                CONCAT(str_EndPoint,',',str_Speed,','),
260                                CONCAT(str_AccelTime,',',str_BlendingValue,','),
261                                CONCAT(str_AcrAngle,',',str_PrecisePositioning,')') );
262
263
264       //When no blending Motion ended acknowledgement with QueueTag()
265       IF NOT(iParameters.BlendingEnable) THEN
266          Script_Command:=CONCAT(Script_Command,'$R$LQueueTag(',str_CommandID,',0)');
267       END_IF;
268
269       //Exits Listen Node in TMflow with ScriptExit()
270       IF iParameters.ExitNode THEN
271          Script_Command:=CONCAT(Script_Command,'$R$L','ScriptExit()');
272       END_IF;
```

Figure 5.3.4.4 – CBT_MoveCircle FB algorithm step 40 building the message with the complete command (source: own).

Once the message has been built, it is needed to find the number of characters of (str_CommandID + Script_Command) and to store it in the string (str_Length). Then, joining all parts (Header + str_Length + str_CommandID + Script_Command) to calculate the checksum (str_Checksum_Calc).

```
274         // CheckSum calculation
275         str_Length:=INT_TO_STRING(LEN(CONCAT(str_CommandID,',',Script_Command)));
276         str_Checksum_Calc := CONCAT(CONCAT(Header,',',str_Length,',',str_CommandID),CONCAT(',',Script_Command),',');
277
278         Checksum_Length:=ToAryByte(str_Checksum_Calc,_eBYTE_ORDER#_LOW_HIGH,Send_Checksum[0]);
279
280         IF Checksum_Length>0 THEN
281             FB_step:=50;
282         ELSE
283             ToError_Sts := TRUE;
284             ToErrorID:=16#40;
285             ToErrorDescrip:='Checksum length not valid';
286         END_IF;
```

Figure 5.3.4.5 – CBT_MoveCircle FB algorithm step 40 building the message with the complete command and finds the number of characters in the string (source: own).

Step 50 calculates with XOR operation the string for the checksum (str_Checksum) and created the complete message to be sent (str_SendFrame) composed by:

- Header
- Length of the message
- Command ID
- String command
- CheckSum calculation

```
290 50: (*  Frame building  to send command to TMflow *)
291
292         FOR i:=1 TO LEN(str_Checksum_Calc) BY 1 DO
293             Send_Checksum[0]:=Send_Checksum[0] XOR Send_Checksum[i];
294         END_FOR;
295
296
297         str_Checksum:=BYTE_TO_STRING(Send_Checksum[0]);
298         str_SendFrame := CONCAT(CONCAT('$$',Header,',',str_Length,','),CONCAT(str_CommandID,',',Script_Command),',*',str_Checksum,'$R$L');
299
300         Length:=LEN(str_SendFrame);
301         Long:=ToAryByte(str_SendFrame, _eBYTE_ORDER#_LOW_HIGH,TCP_Send_Data[0]);
302
303         IF Long>0 THEN
304             FB_step:=60;
305         ELSE
306             ToError_Sts := TRUE;
307             ToErrorID:=16#50;
308             ToErrorDescrip:='Final frame building is error end';
309         END_IF;
```

Figure 5.3.4.6 – CBT_MoveCircle FB algorithm step 50 building the complete message (source: own).

From step 60 to the end of the algorithm, refer to the section *5.3.2. CBT_MoveAbsolute*.

Error list with the Error ID, Error Description and Action:

Error ID	Error Description	Action
10	Robot not connected	Use CBT_Connect FB for connection
11	Other FB is under execution	Wait until other FB is done
20	Clear buffer error	Check Ethernet connection wiring
25	Speed range in Line: 0~4500 mm/s	Set parameter in acceptable range
26	Speed range in PTP: 0~100 %	Set parameter in acceptable range
27	Acceleration Time range is: 150~9999 ms	Set parameter in acceptable range
28	Command ID range is: 2~9	Set parameter in acceptable range
40	Checksum length not valid	Re-execution is required
50	Final frame building is error end	Re-execution is required
60	TCP send error	Check Ethernet connection wiring
70	TCP receive error	Check Ethernet connection wiring
80	Command rejected	Re-execution is required
90	Clear buffer error	Check Ethernet connection wiring
100	TCP receive error	Check Ethernet connection wiring
110	Motion failed	Re-execution is required

Table 5.3.4.6 – CBT_MoveCircle error list description and action (source: own).

5.3.5. CBT_ChangeBase

This Function Block sends the command for changing the base coordinates of the follow-up motions into buffer for execution.

Among other parameters, user must set the Cobot In/out variable to identify the target robot with its IP address and communications port, the command ID to identify the command in the feedback sent by the robot and the base coordinates: X, Y and Z in millimetres for positioning, and RX, RY and RZ in degrees for orientation.

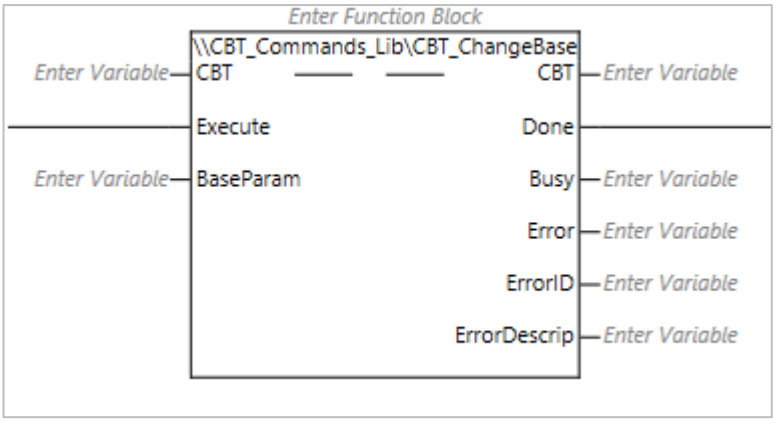

Figure 5.3.5.1 – CBT_ChangeBase Function Block (source: own).

In the below tables, input and output parameters are described:

Name	I/O	Data Type	Valid Range	Default	Description
CBT	In/Out	CBT_Commands_Lib\stCBT_Connection	---	---	Identifies robot IP address, port and connection.
Execute	Input	BOOL	TRUE or FALSE	FALSE	Request of instruction execution
Parameters	Input	CBT_Commands_Lib\stCBT_BaseParam	---	---	Base Parameters
Done	Output	BOOL	TRUE or FALSE	FALSE	TRUE when the instruction is completed
Busy	Output	BOOL	TRUE or FALSE	FALSE	TRUE when the instruction is in progress
Error	Output	BOOL	TRUE or FALSE	FALSE	TRUE while there is an error
ErrorID	Output	WORD	---	16#0000	Contains the error code when an error occurs. A value of 16#0000 indicates normal execution
ErrorDescrip	Output	STRING[50]	---	---	Contains the error description when an error occurs

Table 5.3.5.1 – CBT_ChangeBase variables type, range, default value and description (source: own).

Name	Data Type	Valid Range	Default	Description
Socket	_sSOCKET	---	---	Socket
Connected	BOOL	TRUE or FALSE	FALSE	TRUE when robot is connected
WaitingReturn	BOOL	TRUE or FALSE	FALSE	TRUE when robot is under control

Table 5.3.5.2 – stCBT_Connection datatype, range, default value and description (source: own).

Name	Data Type	Valid Range	Default	Description
CommandID	UINT	2 to 9	0	Identification for the return message for this command
Transformation	CBT_Commands_Lib\stCBT_Transformation	---	---	Base coordinates
ExitNode	BOOL	TRUE or FALSE	FALSE	Quit Listen node in robot program

Table 5.3.5.3 – stCBT_BaseParam datatype, range, default value and description (source: own).

Name	Data Type	Valid Range	Default	Description
X	REAL	---	0	X coordinates in mm
Y	REAL	---	0	Y coordinates in mm
Z	REAL	---	0	Z coordinates in mm
RX	REAL	---	0	RX coordinates in degrees
RY	REAL	---	0	RY coordinates in degrees
RZ	REAL	---	0	RZ coordinates in degrees

Table 5.3.5.4 – stCBT_Transformation datatype, range, default value and description (source: own).

Function Block Script

As all Function Blocks in this Library, once the upward signal differentiation of *Execute* input is detected and it is not in *Busy* state, the sequence initialization is started, otherwise an error is triggered. Sequence initialization has been described in *5.3.1.CBT_Connect* section.

As all Function Blocks in this Library, State Diagram Control is part of the script focused on output variable operation and timing. State Diagram Control has been described in *5.3.1.CBT_Connect* section.

From step 0 to 30 and from step 60 to the end, refer to the section *5.3.2. CBT_MoveAbsolute*.

Hereunder, the description of the specific script of CBT_ChangeBase FB:

Step 30 converts base transformation parameter from real to string (str_TargetPosition).

```
30: (*  iParam conversion to string *)

    str_CommandID:=UINT_TO_STRING(iParameters.CommandID);

    str_TP_X := RealToFormatString(In:=iParameters.Transformation.X, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
    str_TP_Y := RealToFormatString(In:=iParameters.Transformation.Y, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
    str_TP_Z := RealToFormatString(In:=iParameters.Transformation.Z, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
    str_TP_RX := RealToFormatString(In:=iParameters.Transformation.RX, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
    str_TP_RY := RealToFormatString(In:=iParameters.Transformation.RY, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
    str_TP_RZ := RealToFormatString(In:=iParameters.Transformation.RZ, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
    str_TP1:=CONCAT(str_TP_X,',',str_TP_Y,',',str_TP_Z);
    str_TP2:=CONCAT(str_TP_RX,',',str_TP_RY,',',str_TP_RZ);
    str_TargetPosition:=CONCAT(str_TP1,',',str_TP2);

    FB_step:=40;
```

Figure 5.3.5.2 – CBT_ChangeBase FB algorithm step 30 creating the string of the base transformation (source: own).

Step 40 builds the message (script_command) to be sent to the robot including:

- Function command
- Base transformation

➢ If ExitNode is enabled, ScriptExit() command is included in the message. It exits external script control mode once the last motion command has been executed and finished. Listen Node is quitted, and not further external script functions commands are accepted.

```
184
185  40: (*  Frame building  for CheckSum calculation *)
186
187          Header:='TMSCT';
188          str_FunctionCommand:='ChangeBase';
189          Script_Command:=CONCAT(str_FunctionCommand,'(',str_TargetPosition,')');
190
191          //Exits Listen Node in TMflow with ScriptExit()
192          IF iParameters.ExitNode THEN
193             Script_Command:=CONCAT(Script_Command,'$R$L','ScriptExit()');
194          END_IF;
```

Figure 5.3.5.3 – CBT_ChangeBase FB algorithm step 40 building the message with the complete command (source: own).

Once the message has been built, it is needed to find the number of characters of (str_CommandID + Script_Command) and to store it in the string (str_Length). Then, joining all parts (Header + str_Length + str_CommandID + Script_Command) to calculate the checksum (str_Checksum_Calc).

```
196          // CheckSum calculation
197          str_Length:=INT_TO_STRING(LEN(CONCAT(str_CommandID,',',Script_Command)));
198          str_Checksum_Calc := CONCAT(CONCAT(Header,',',str_Length,',',str_CommandID),CONCAT(',',Script_Command),',');
199
200          Checksum_Length:=ToAryByte(str_Checksum_Calc,_eBYTE_ORDER#_LOW_HIGH,Send_Checksum[0]);
201
202          IF Checksum_Length>0 THEN
203             FB_step:=50;
204          ELSE
205             ToError_Sts := TRUE;
206             ToErrorID:=16#40;
207             ToErrorDescrip:='Checksum length not valid';
208          END_IF;
```

Figure 5.3.5.4 – CBT_ChangeBase FB algorithm step 40 building the message with the complete command and finds the number of characters in the string (source: own).

Step 50 calculates with XOR operation the string for the checksum (str_Checksum) and created the complete message to be sent (str_SendFrame) composed by:

- Header
- Length of the message
- Command ID
- String command
- CheckSum calculation

```
212  50: (*  Frame building  to send command to TMflow *)
213
214      FOR i:=1 TO LEN(str_Checksum_Calc) BY 1 DO
215         Send_Checksum[0]:=Send_Checksum[0] XOR Send_Checksum[i];
216      END_FOR;
217
218      str_Checksum:=BYTE_TO_STRING(Send_Checksum[0]);
219      str_SendFrame := CONCAT(CONCAT('$$',Header,',',str_Length,','),CONCAT(str_CommandID,',',Script_Command),',*',str_Checksum,'$R$L');
220
221      Length:=LEN(str_SendFrame);
222      Long:=ToAryByte(str_SendFrame, _eBYTE_ORDER#_LOW_HIGH,TCP_Send_Data[0]);
223
224      IF Long>0 THEN
225         FB_step:=60;
226      ELSE
227         ToError_Sts := TRUE;
228         ToErrorID:=16#50;
229         ToErrorDescrip:='Final frame building is error end';
230      END_IF;
231
```

Figure 5.3.5.5 – CBT_ChangeBase FB algorithm step 50 building the complete message (source: own).

From step 60 to the end of the algorithm is already described in previous FB explanation.

Error list with the Error ID, Error Description and Action:

Error ID	Error Description	Action
10	Robot not connected	Use CBT_Connect FB for connection
11	Other FB is under execution	Wait until other FB is done
20	Clear buffer error	Check Ethernet connection wiring
40	Checksum length not valid	Re-execution is required
50	Final frame building is error end	Re-execution is required
60	TCP send error	Check Ethernet connection wiring
70	TCP receive error	Check Ethernet connection wiring
80	Command rejected	Re-execution is required

Table 5.3.5.5 – CBT_ChangeBase error list description and action (source: own).

5.3.6. CBT_ChangeTCP

This Function Block sends the command for changing the TCP offset value of the follow-up motions into buffer for execution.

Among other parameters, user must set the Cobot In/out variable to identify the target robot with its IP address and communications port, the command ID to identify the command in the feedback sent by the robot, the base coordinates (X, Y and Z in millimetres for positioning, and RS, RY and RZ in degrees for orientation), the weight (in kilograms), the moment of inertia (Ixx, Iyy and Izz in kg·mm²), and the location of the mass center (X, Y and Z in millimetres for positioning, and RX, RY and RZ in degrees for orientation).

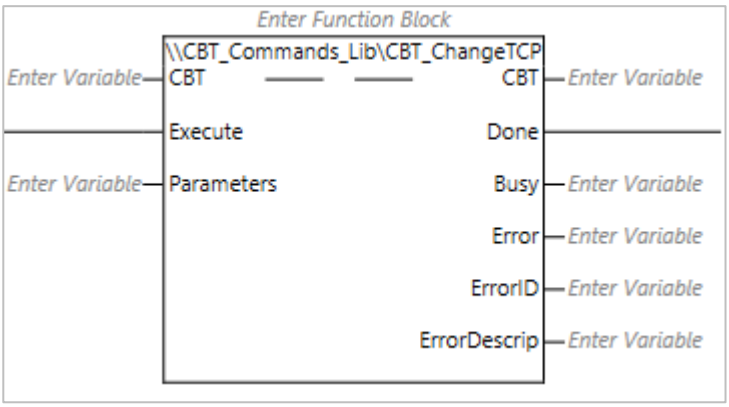

Figure 5.3.6.1 – CBT_ChangeTCP Function Block (source: own).

In the below tables, input and output parameters are described:

Name	I/O	Data Type	Valid Range	Default	Description
CBT	In/Out	CBT_Commands_Lib\stCBT_Connection	---	---	Identifies robot IP address, port and connection.
Execute	Input	BOOL	TRUE or FALSE	FALSE	Request of instruction execution
Parameters	Input	CBT_Commands_Lib\stCBT_TCPParam	---	---	TCP parameters
Done	Output	BOOL	TRUE or FALSE	FALSE	TRUE when the instruction is completed
Busy	Output	BOOL	TRUE or FALSE	FALSE	TRUE when the instruction is in progress
Error	Output	BOOL	TRUE or FALSE	FALSE	TRUE while there is an error
ErrorID	Output	WORD	---	16#0000	Contains the error code when an error occurs. A value of 16#0000 indicates normal execution
ErrorDescrip	Output	STRING[50]	---	---	Contains the error description when an error occurs

Table 5.3.6.1 – CBT_ChangeTCP variables type, range, default value and description (source: own).

Name	Data Type	Valid Range	Default	Description
Socket	_sSOCKET	---	---	Socket
Connected	BOOL	TRUE or FALSE	FALSE	TRUE when robot is connected
WaitingReturn	BOOL	TRUE or FALSE	FALSE	TRUE when robot is under control

Table 5.3.6.2 – stCBT_Connection datatype, range, default value and description (source: own).

Name	Data Type	Valid Range	Default	Description
CommandID	UINT	2 to 9	0	Identification for the return message for this command
TCPOffset	CBT_Commands_Lib\stCBT_Transformation	---	---	Tool Offset coordinates
Weight	UINT	0 to 14	0	Weight of the tool
MomentOfInertia	CBT_Commands_Lib\stCBT_MomentOfInertia	---	---	Tool's moment of inertia
MassCenter	CBT_Commands_Lib\stCBT_Transformation	---	---	Frame reference of mass center
ExitNode	BOOL	TRUE or FALSE	FALSE	Quit Listen node in robot program

Table 5.3.6.3 – stCBT_ TCPParam datatype, range, default value and description (source: own).

Name	Data Type	Valid Range	Default	Description
X	REAL	---	0	X coordinates in mm
Y	REAL	---	0	Y coordinates in mm
Z	REAL	---	0	Z coordinates in mm
RX	REAL	---	0	RX coordinates in degrees
RY	REAL	---	0	RY coordinates in degrees
RZ	REAL	---	0	RZ coordinates in degrees

Table 5.3.6.4 – stCBT_Transformation datatype, range, default value and description (source: own).

Name	Data Type	Valid Range	Default	Description
Ixx	REAL	---	0	Moment of inertia around X in kg·mm²
Iyy	REAL	---	0	Moment of inertia around Y in kg·mm²
Izz	REAL	---	0	Moment of inertia around Z in kg·mm²

Table 5.3.6.5 – stCBT_ MomentOfInertia datatype, range, default value and description (source: own).

Function Block Script

As all Function Blocks in this Library, once the upward signal differentiation of *Execute* input is detected and it is not in *Busy* state, the sequence initialization is started, otherwise an error is triggered. Sequence initialization has been described in *5.3.1.CBT_Connect* section.

As all Function Blocks in this Library, State Diagram Control is part of the script focused on output variable operation and timing. State Diagram Control has been described in *5.3.1.CBT_Connect* section.

From step 0 to 30 and from step 60 to the end, refer to the section *5.3.2. CBT_MoveAbsolute*.

Hereunder, the description of the specific script of CBT_ChangeTCP FB:

Step 30 converts TCP transformation offset parameter from real to string (str_TCPOffset), Inertia parameter from real to string (str_Inertia), Weight parameter from real to string (str_Weight), Mass Center parameter from real to string (str_MassCenter).

```
30: (*  iBaseParam conversion to string *)

        str_CommandID:=UINT_TO_STRING(iParameters.CommandID);

        //TCPOffset
        str_TO_X := RealToFormatString(In:=iParameters.TCPOffset.X, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_TO_Y := RealToFormatString(In:=iParameters.TCPOffset.Y, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_TO_Z := RealToFormatString(In:=iParameters.TCPOffset.Z, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_TO_RX := RealToFormatString(In:=iParameters.TCPOffset.RX, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_TO_RY := RealToFormatString(In:=iParameters.TCPOffset.RY, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_TO_RZ := RealToFormatString(In:=iParameters.TCPOffset.RZ, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_TO1:=CONCAT(str_TO_X,',',str_TO_Y,',',str_TO_Z);
        str_TO2:=CONCAT(str_TO_RX,',',str_TO_RY,',',str_TO_RZ);
        str_TCPOffset:=CONCAT(str_TO1,',',str_TO2);

        //MomentOfInertia
        str_Ixx := RealToFormatString(In:=iParameters.MomentOfInertia.Ixx, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_Iyy := RealToFormatString(In:=iParameters.MomentOfInertia.Iyy, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_Izz := RealToFormatString(In:=iParameters.MomentOfInertia.Izz, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_Inertia:=CONCAT(str_Ixx,',',str_Iyy,',',str_Izz);

        //Weight
        str_Weight := RealToFormatString(In:=iParameters.Weight, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);

        //MassCenter
        str_MC_X := RealToFormatString(In:=iParameters.MassCenter.X, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_MC_Y := RealToFormatString(In:=iParameters.MassCenter.Y, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_MC_Z := RealToFormatString(In:=iParameters.MassCenter.Z, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_MC_RX := RealToFormatString(In:=iParameters.MassCenter.RX, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_MC_RY := RealToFormatString(In:=iParameters.MassCenter.RY, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_MC_RZ := RealToFormatString(In:=iParameters.MassCenter.RZ, Exponent:=FALSE, Sign:=TRUE, MinLen:=1, DecPlace:=0);
        str_MC1:=CONCAT(str_MC_X,',',str_MC_Y,',',str_MC_Z);
        str_MC2:=CONCAT(str_MC_RX,',',str_MC_RY,',',str_MC_RZ);
        str_MassCenter:=CONCAT(str_MC1,',',str_MC2);

        str_TCP1:=CONCAT(str_TCPOffset,',',str_Weight,',');
        str_TCP2:=CONCAT(str_Inertia,',',str_MassCenter);
        str_TCPCommand:=CONCAT(str_TCP1,str_TCP2);

        FB_step:=40;
```

Figure 5.3.6.2 – CBT_ChangeTCP FB algorithm step 30 creating the string of the complete command (source: own).

Step 40 builds the message (script_command) to be sent to the robot including:

- Function command
- TCP offset

➢ If ExitNode is enabled, ScriptExit() command is included in the message. It exits external script control mode once the last motion command has been executed and finished. Listen Node is quitted, and not further external script functions commands are accepted.

```
219  40: (*  Frame building for CheckSum calculation *)
220
221      Header:='TMSCT';
222      str_FunctionCommand:='ChangeTCP';
223      Script_Command:=CONCAT(str_FunctionCommand,'(',str_TCPCommand,')');
224
225      //Exits Listen Node in TMflow with ScriptExit()
226      IF iParameters.ExitNode THEN
227         Script_Command:=CONCAT(Script_Command,'$R$L','ScriptExit()');
228      END_IF;
229
```

Figure 5.3.6.3 – CBT_ChangeTCP FB algorithm step 40 building the message with the complete command (source: own).

Once the message has been built, it is needed to find the number of characters of (str_CommandID + Script_Command) and to store it in the string (str_Length). Then, joining all parts (Header + str_Length + str_CommandID + Script_Command) to calculate the checksum (str_Checksum_Calc).

```
230      // CheckSum calculation
231      str_Length:=INT_TO_STRING(LEN(CONCAT(str_CommandID,',',Script_Command)));
232      str_Checksum_Calc := CONCAT(CONCAT(Header,',',str_Length,',',str_CommandID),CONCAT(',',Script_Command),',');
233
234      Checksum_Length:=ToAryByte(str_Checksum_Calc,_eBYTE_ORDER#_LOW_HIGH,Send_Checksum[0]);
235
236      IF Checksum_Length>0 THEN
237         FB_step:=50;
238      ELSE
239         ToError_Sts := TRUE;
240         ToErrorID:=16#40;
241         ToErrorDescrip:='Checksum length not valid';
242      END_IF;
243
```

Figure 5.3.6.4 – CBT_ChangeTCP FB algorithm step 40 building the message with the complete command and finds the number of characters in the string (source: own).

Step 50 calculates with XOR operation the string for the checksum (str_Checksum) and created the complete message to be sent (str_SendFrame) composed by:

- Header
- Length of the message
- Command ID
- String command
- CheckSum calculation

```
50: (*  Frame building  to send command to TMflow *)

    FOR i:=1 TO LEN(str_Checksum_Calc) BY 1 DO
       Send_Checksum[0]:=Send_Checksum[0] XOR Send_Checksum[i];
    END_FOR;


    str_Checksum:=BYTE_TO_STRING(Send_Checksum[0]);
    str_SendFrame := CONCAT(CONCAT('$$',Header,',',str_Length,','),CONCAT(str_CommandID,',',Script_Command),',*',str_Checksum,'$R$L');

    Length:=LEN(str_SendFrame);
    Long:=ToAryByte(str_SendFrame, _eBYTE_ORDER#_LOW_HIGH,TCP_Send_Data[0]);

    IF Long>0 THEN
       FB_step:=60;
    ELSE
       ToError_Sts := TRUE;
       ToErrorID:=16#50;
       ToErrorDescrip:='Final frame building is error end';
    END_IF;
```

Figure 5.3.6.5 – CBT_ChangeTCP FB algorithm step 50 building the complete message (source: own).

From step 60 to the end of the algorithm, refer to the section *5.3.2. CBT_MoveAbsolute*.

Error list with the Error ID, Error Description and Action:

Error ID	Error Description	Action
10	Robot not connected	Use CBT_Connect FB for connection
11	Other FB is under execution	Wait until other FB is done
20	Clear buffer error	Check Ethernet connection wiring
25	Weight range: 0~14 kg	Set parameter in acceptable range
40	Checksum length not valid	Re-execution is required
50	Final frame building is error end	Re-execution is required
60	TCP send error	Check Ethernet connection wiring
70	TCP receive error	Check Ethernet connection wiring
80	Command rejected	Re-execution is required

Table 5.3.6.6 – CBT_ChangeTCP error list description and action (source: own).

5.3.7. CBT_ProgramControl

This Function Block sends the command to the robot to pause or resume the program execution. If the program is paused during robot movement, the robot will complete the interrupted movement once the program execution is restored.

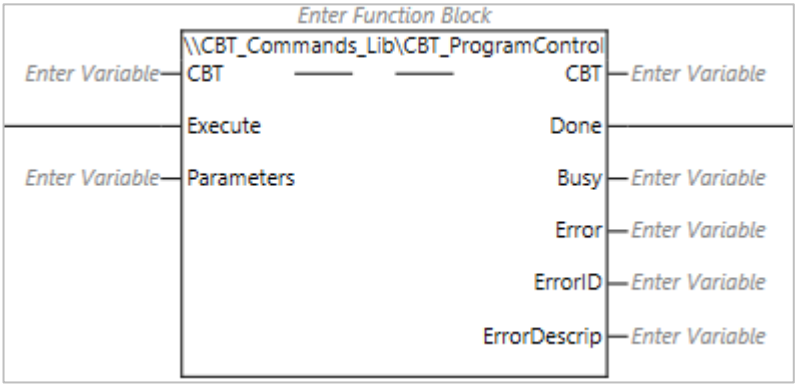

Figure 5.3.7.1 – CBT_ProgramControl Function Block (source: own).

In the below tables, input and output parameters are described:

Name	I/O	Data Type	Valid Range	Default	Description
CBT	In/Out	CBT_Commands_Lib\stCBT_Connection	---	---	Identifies robot IP address, port and connection.
Execute	Input	BOOL	TRUE or FALSE	FALSE	Request of instruction execution
Parameters	Input	CBT_Commands_Lib\stCBT_PrgControl	---	---	Program control parameters
Done	Output	BOOL	TRUE or FALSE	FALSE	TRUE when the instruction is completed
Busy	Output	BOOL	TRUE or FALSE	FALSE	TRUE when the instruction is in progress
Error	Output	BOOL	TRUE or FALSE	FALSE	TRUE while there is an error
ErrorID	Output	WORD	---	16#0000	Contains the error code when an error occurs. A value of 16#0000 indicates normal execution
ErrorDescrip	Output	STRING[50]	---	---	Contains the error description when an error occurs

Table 5.3.7.1 – CBT_ProgramControl variables, range, default value and description (source: own).

Name	Data Type	Valid Range	Default	Description
Socket	_sSOCKET	---	---	Socket
Connected	BOOL	TRUE or FALSE	FALSE	TRUE when robot is connected
WaitingReturn	BOOL	TRUE or FALSE	FALSE	TRUE when robot is under control

Table 5.3.7.2 – stCBT_Connection datatype, range, default value and description (source: own).

Name	Data Type	Valid Range	Default	Description
CommandID	UINT	2 to 9	0	Identification for the return message for this command
Command	CBT_Commands_Lib\eCBT_PrgControlCmd	---	---	Pause or resume program

Table 5.3.7.3 – stCBT_PrgControl datatype, range, default value and description (source: own).

Name	Enum Value	Description
Pause	0	Pause the project and the motion of the robot
Resume	1	Resume the project and the motion of the robot

Table 5.3.7.4 – eCBT_PrgControlCmd datatype description (source: own).

Function Block Script

As all Function Blocks in this Library, once the upward signal differentiation of *Execute* input is detected and it is not in *Busy* state, the sequence initialization is started, otherwise an error is triggered. Sequence initialization has been described in *5.3.1.CBT_Connect* section.

As all Function Blocks in this Library, State Diagram Control is part of the script focused on output variable operation and timing. State Diagram Control has been described in *5.3.1.CBT_Connect* section.

From step 0 to 30 and from step 60 to the end, refer to the section *5.3.2. CBT_MoveAbsolute*.

Hereunder, the description of the specific script of CBT_ProgramControl FB:

Step 30 converts Program Control Command parameter from enumerated to string (str_FunctionCommand).

```
30: (*  iParam conversion to string *)

    str_CommandID:=UINT_TO_STRING(iParameters.CommandID);

    // iParameters.MovementCommand
    IF iParameters.Command=\\CBT_Commands_Lib\eCBT_PrgControlCmd#Pause THEN
       str_FunctionCommand:='Pause()';
    END_IF;

    IF iParameters.Command=\\CBT_Commands_Lib\eCBT_PrgControlCmd#Resume THEN
       str_FunctionCommand:='Resume()';
    END_IF;

    FB_step:=40;
```

Figure 5.3.7.2 – CBT_ProgramControl FB algorithm step 30 creating the string of the complete command (source: own).

Step 40 builds the message (script_command) to be sent to the robot including:

- Function command

```
40: (*  Frame building  for CheckSum calculation *)

    Header:='TMSCT';

    Script_Command:=str_FunctionCommand;
```

Figure 5.3.7.3 – CBT_ProgramControl FB algorithm step 40 building the message with the complete command (source: own).

Once the message has been built, it is needed to find the number of characters of (str_CommandID + Script_Command), and to store it in the string (str_Length). Then, joining all parts (Header + str_Length + str_CommandID + Script_Command) to calculate the checksum (str_Checksum_Calc).

```
185
186         // CheckSum calculation
187         str_Length:=INT_TO_STRING(LEN(CONCAT(str_CommandID,',',Script_Command)));
188         str_Checksum_Calc := CONCAT(CONCAT(Header,',',str_Length,',',str_CommandID),CONCAT(',',Script_Command),',');
189
190         Checksum_Length:=ToAryByte(str_Checksum_Calc,_eBYTE_ORDER#_LOW_HIGH,Send_Checksum[0]);
191
192         IF Checksum_Length>0 THEN
193             FB_step:=50;
194         ELSE
195             ToError_Sts := TRUE;
196             ToErrorID:=16#40;
197             ToErrorDescrip:='Checksum length not valid';
198         END_IF;
199
```

Figure 5.3.7.4 – CBT_ProgramControl FB algorithm step 40 building the message with the complete command and finds the number of characters in the string (source: own).

Step 50 calculates with XOR operation the string for the checksum (str_Checksum) and created the complete message to be sent (str_SendFrame) composed by:

- Header
- Length of the message
- Command ID
- String command
- CheckSum calculation

```
201
202 50: (*  Frame building  to send command to TMflow *)
203
204         FOR i:=1 TO LEN(str_Checksum_Calc) BY 1 DO
205             Send_Checksum[0]:=Send_Checksum[0] XOR Send_Checksum[i];
206         END_FOR;
207
208         str_Checksum:=BYTE_TO_STRING(Send_Checksum[0]);
209         str_SendFrame := CONCAT(CONCAT('$$',Header,',',str_Length,','),CONCAT(str_CommandID,',',Script_Command),',*',str_Checksum,'$R$L');
210
211         Length:=LEN(str_SendFrame);
212         Long:=ToAryByte(str_SendFrame, _eBYTE_ORDER#_LOW_HIGH,TCP_Send_Data[0]);
213
214         IF Long>0 THEN
215             FB_step:=60;
216         ELSE
217             ToError_Sts := TRUE;
218             ToErrorID:=16#50;
219             ToErrorDescrip:='Final frame building is error end';
220         END_IF;
221
```

Figure 5.3.7.5 – CBT_ProgramControl FB algorithm step 50 building the complete message (source: own).

From step 60 to the end of the algorithm, refer to the section *5.3.2. CBT_MoveAbsolute*.

Error list with the Error ID, Error Description and Action:

Error ID	Error Description	Action
10	Robot not connected	Use CBT_Connect FB for connection
20	Clear buffer error	Check Ethernet connection wiring
40	Checksum length not valid	Re-execution is required
50	Final frame building is error end	Re-execution is required
60	TCP send error	Check Ethernet connection wiring
70	TCP receive error	Check Ethernet connection wiring
80	Command rejected	Re-execution is required

Table 5.3.7.5 – CBT_ProgramControl error list description and action (source: own).

5.4. Program example

A program example in Ladder language (IEC 61131-3 language) has been created to execute each Function Block independently. Each rung contains: Execute input contact, FB instance and Done output relay. Almost all variables have been declared as Global Variables to be reachable from and HMI.

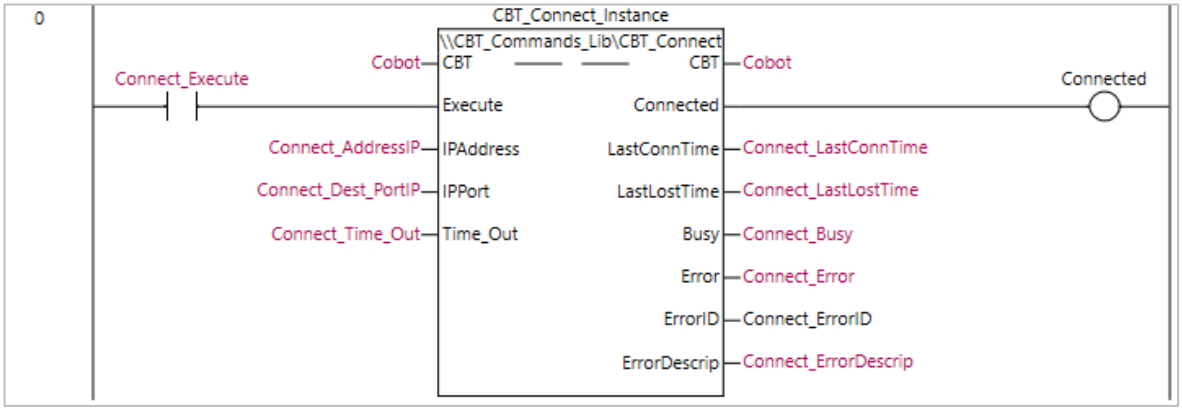

Figure 5.4.1 – CBT_Connect FB in the program example (source: own).

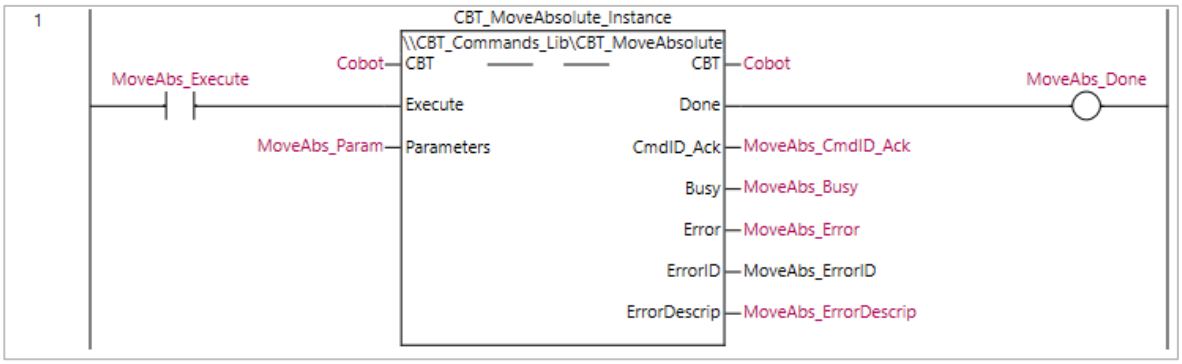

Figure 5.4.2 – CBT_MoveAbsolute FB in the program example (source: own).

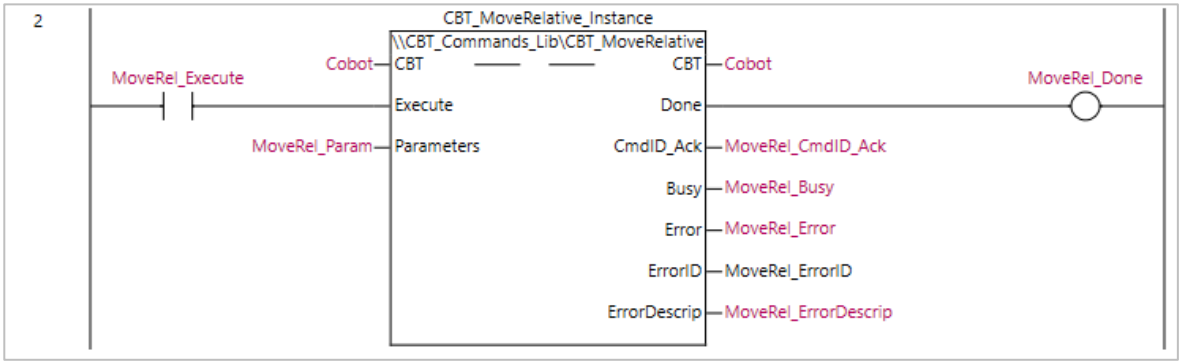

Figure 5.4.3 – CBT_MoveRelative FB in the program example (source: own).

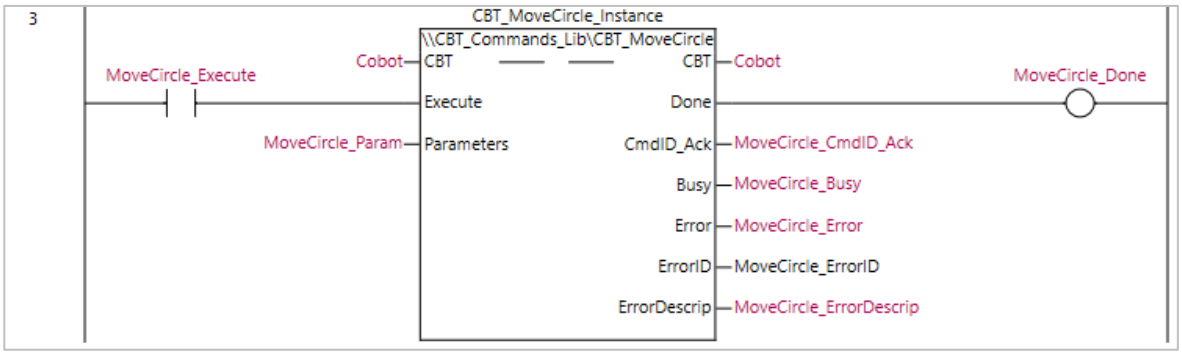

Figure 5.4.4 – CBT_MoveCircle FB in the program example (source: own).

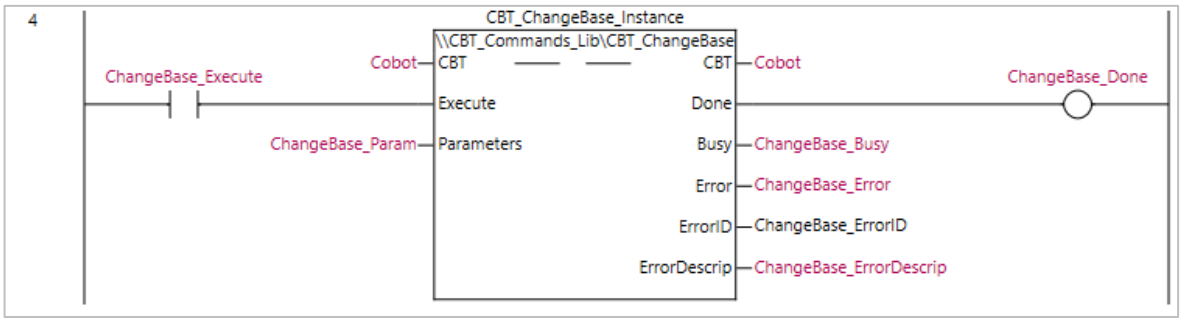

Figure 5.4.5 – CBT_ChangeBase FB in the program example (source: own).

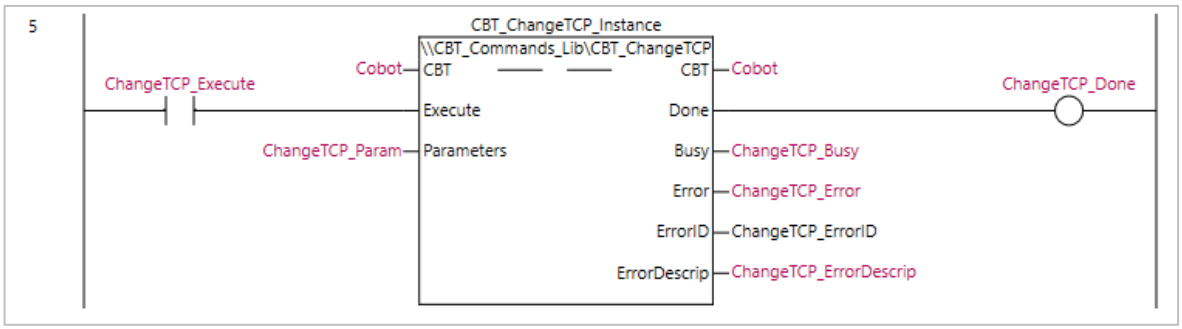

Figure 5.4.6 – CBT_ChangeTCP FB in the program example (source: own).

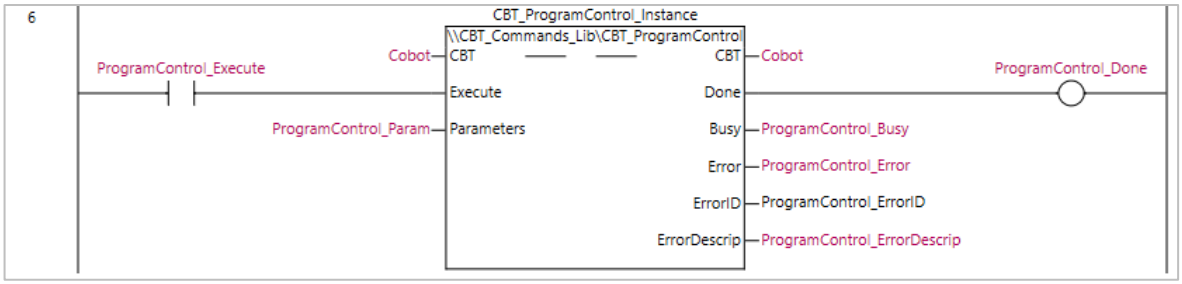

Figure 5.4.7 – CBT_ProgramControl FB in the program example (source: own).

A second program has been created for a PnP (Pick and Place) application in which only the CBT_MoveAbsolute has been used. For this, the program is composed by 6 steps, each one corresponds to one of the 6 segments of the typical PnP trajectory cycle (see the whole program in the Annex *B1. Pick and Place sequence*).

On this program example, Cobot in/out variable identifying the target robot with its IP address and communications port must be the same one. Then, for each one of the 6 CBT_MoveAbsolute instances execution, it is possible to set the command ID to identify the command in the feedback sent by the robot in case there is no blending, speed including units, acceleration time, if blending is required and if it is performed by percentage or by radius and the robot arm configuration for PTP movements.

In the image below it can be seen the trajectory of the robot TCP (Tool Center Point) description for a typical PnP cycle.

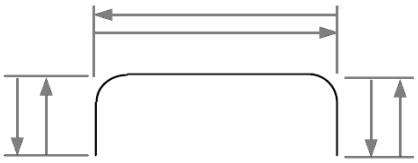

Figure 5.4.8 – PnP cycle (source: own).

6. HMI

6.1. Description

The target of developing this GUI (Graphical User Interface) is providing to the user a simple graphical environment to easily use the Function Blocks and send commands to the robot. The project has been designed defining a Main page as primary menu giving direct access to secondary pages with different purposes:

- Get connection with robot.
- Usage of all function and motion command Function Blocks.
- PnP application example.

Sysmac Studio is the software used to do the programming of the HMI. It uses VB.NET (Visual Basic .NET) object-oriented programming language implemented in .NET Framework. Most of the graphical components are configured using element properties like appearance (text font, margins, colours, visibility), behaviour (variable associated with maximum and minimum values, display format, availability, etc), layout (position and size) and security (access level or visibility level). Events and action can be set when pressing or releasing a button (buzzer, call a routine of VB programming, open a page, reset a variable, etc.).

6.2. Mapping Variables

Mapping variables refers to assigning global variables in the PLC connected to the HMI to global variables in the HMI. Therefore, mapping variables is required to link input value from user in the HMI with the input parameters in the FBs.

PLC Variables	DataType	HMI Variable
ChangeBase_Busy	BOOL	HMI_ChangeBase_Busy
ChangeBase_Done	BOOL	HMI_ChangeBase_Done
ChangeBase_Error	BOOL	HMI_ChangeBase_Error
ChangeBase_ErrorDescrip	STRING[50]	HMI_ChangeBase_ErrorDescrip
ChangeBase_Execute	BOOL	HMI_ChangeBase_Execute
ChangeBase_Param	CBT_Commands_Lib\stCBT_BaseParam	HMI_ChangeBase_Parameters
ChangeTCP_Busy	BOOL	HMI_ChangeTCP_Busy

ChangeTCP_Done	BOOL	HMI_ChangeTCP_Done
ChangeTCP_Error	BOOL	HMI_ChangeTCP_Error
ChangeTCP_ErrorDescrip	STRING[50]	HMI_ChangeTCP_ErrorDescrip
ChangeTCP_Execute	BOOL	HMI_ChangeTCP_Execute
ChangeTCP_Param	CBT_Commands_Lib\stCBT_TCPParam	HMI_ChangeTCP_Parameters
Cobot	CBT_Commands_Lib\stCBT_Connection	HMI_Cobot
Connect_AddressIP	STRING[50]	HMI_Connect_AddressIP
Connect_Busy	BOOL	HMI_Connect_Busy
Connect_Dest_PortIP	INT	HMI_Connect_Dest_PortIP
Connect_Error	BOOL	HMI_Connect_Error
Connect_ErrorDescrip	STRING[50]	HMI_Connect_ErrorDescrip
Connect_Execute	BOOL	HMI_Connect_Execute
Connect_LastConnTime	DATE_AND_TIME	HMI_Connect_LastConnTime
Connect_LastLostTime	DATE_AND_TIME	HMI_Connect_LastLostTime
Connect_Time_Out	UINT	HMI_Connect_Time_Out
MoveAbs_Busy	BOOL	HMI_MoveAbs_Busy
MoveAbs_CmdID_Ack	UINT	HMI_MoveAbs_CmdID_Ack
MoveAbs_Done	BOOL	HMI_MoveAbs_Done
MoveAbs_Error	BOOL	HMI_MoveAbs_Error
MoveAbs_ErrorDescrip	STRING[50]	HMI_MoveAbs_ErrorDescrip
MoveAbs_Execute	BOOL	HMI_MoveAbs_Execute
MoveAbs_Param	CBT_Commands_Lib\stCBT_MovAbsParam	HMI_MoveAbs_Parameters
MoveCircle_Busy	BOOL	HMI_MoveCircle_Busy
MoveCircle_CmdID_Ack	UINT	HMI_MoveCircle_CmdID_Ack
MoveCircle_Done	BOOL	HMI_MoveCircle_Done
MoveCircle_Error	BOOL	HMI_MoveCircle_Error
MoveCircle_ErrorDescrip	STRING[50]	HMI_MoveCircle_ErrorDescrip
MoveCircle_Execute	BOOL	HMI_MoveCircle_Execute
MoveCircle_Param	CBT_Commands_Lib\stCBT_MovCircleParam	HMI_MoveCircle_Parameters
MoveRel_Busy	BOOL	HMI_MoveRel_Busy
MoveRel_CmdID_Ack	UINT	HMI_MoveRel_CmdID_Ack
MoveRel_Done	BOOL	HMI_MoveRel_Done
MoveRel_Error	BOOL	HMI_MoveRel_Error
MoveRel_ErrorDescrip	STRING[50]	HMI_MoveRel_ErrorDescrip
MoveRel_Execute	BOOL	HMI_MoveRel_Execute
MoveRel_Param	CBT_Commands_Lib\stCBT_MovRelParam	HMI_MoveRel_Parameters
PnP_Example_Execute	BOOL	HMI_PnP_Example_Execute
PnP_Move_CmdID_Ack	UINT	HMI_PnP_Move_CmdID_Ack
PnP_MoveBusy	BOOL	HMI_PnP_MoveBusy
PnP_MoveDone	BOOL	HMI_PnP_MoveDone
PnP_MoveError	BOOL	HMI_PnP_MoveError
PnP_MoveErrorDescrip	STRING[50]	HMI_PnP_MoveErrorDescrip
PnP_MoveParameters	ARRAY[0..5] OF CBT_Commands_Lib\stCBT_MovAbsParam	HMI_PnP_MoveParameters
ProgramControl_Busy	BOOL	HMI_ProgramControl_Busy
ProgramControl_Done	BOOL	HMI_ProgramControl_Done
ProgramControl_Error	BOOL	HMI_ProgramControl_Error
ProgramControl_ErrorDescrip	STRING[50]	HMI_ProgramControl_ErrorDescrip
ProgramControl_Execute	BOOL	HMI_ProgramControl_Execute
ProgramControl_Param	CBT_Commands_Lib\stCBT_PrgControl	HMI_ProgramControl_Parameters

Table 6.2.1 – PLC – HMI variables mapping (source: own).

6.3. Pages

Pages are described in detail in the following sections, but here a picture of the whole navigation tree is shown.

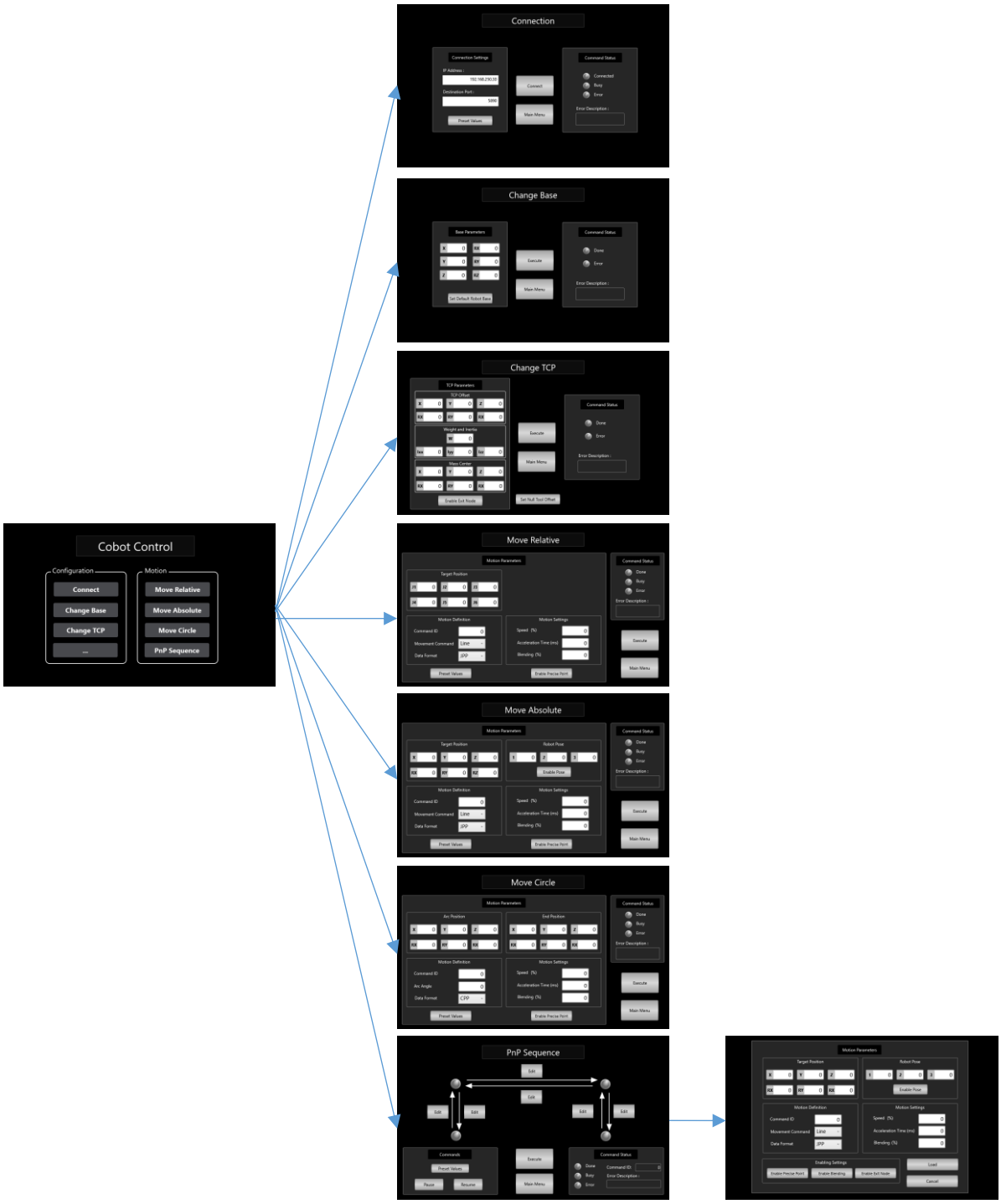

Figure 6.3.0 – HMI navigation tree (source: own).

6.3.1. Main

Once the touchscreen boots up, this is the initial page. It gives access to all the secondary pages which independently handles each FB in the library. As shown in the picture below, it is composed by 2 areas: Configuration and Motion.

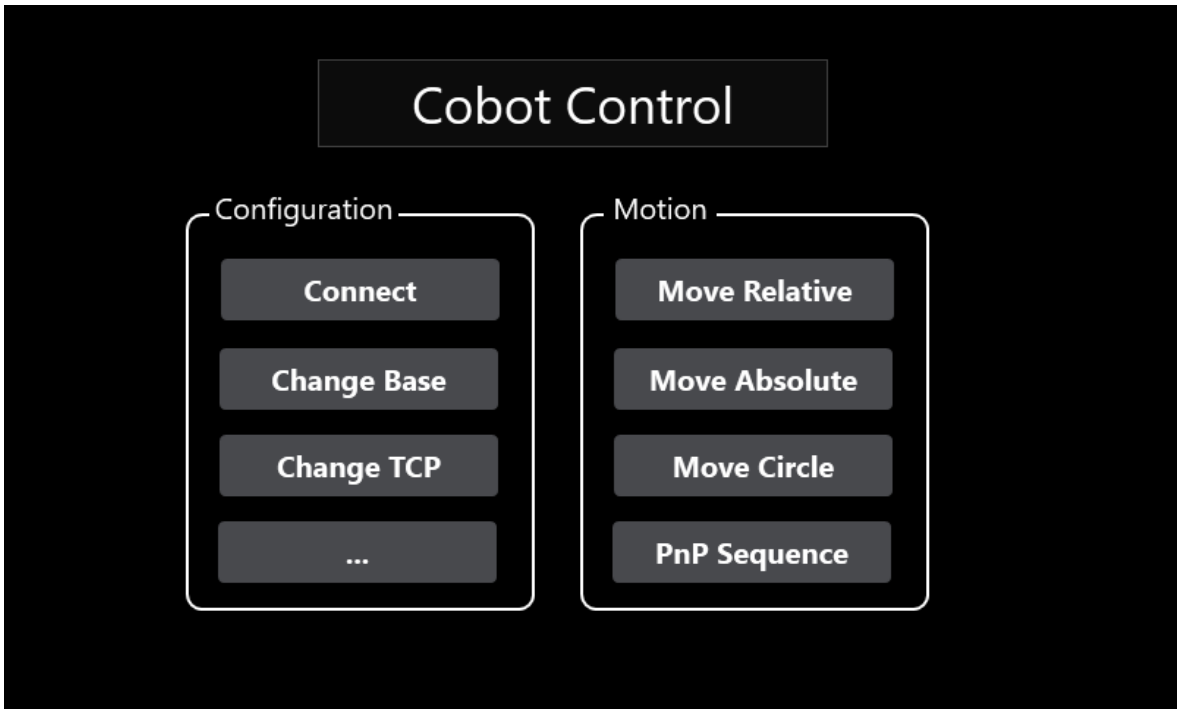

Figure 6.3.1 – Main page of HMI (source: own).

Configuration area includes the actions most needed to start moving the robot, connection with controller, base coordinates, and TCP settings:

- *Connect* button. Opens the page in which the connection with the robot can be done. Variables are mapped to the *CBT_Connect* Function Block in the program example (*Figure 5.4.1*).
- *Change Base* button. Opens the page in which the base coordinate can be set for the follow-up motions into buffer for execution. Variables are mapped to the *CBT_ChangeBase* Function Block in the program example (*Figure 5.4.5*).
- *Change TCP* button. Opens the page in which the TCP coordinates, weight, inertia, and mass center can be configured for the follow-up motions into buffer for execution. Variables are mapped to the *CBT_ChangeTCP* Function Block in the program example (*Figure 5.4.6*).
- A fourth button is reserved for future implementations.

Motion area includes the different motion commands and a Pick and Place application program sample.

- *Move Relative* button. Opens the page in which the robot can be commanded to a relative position defining the motion parameters. Variables are mapped to the *CBT_MoveRelative* Function Block in the program example (*Figure 5.4.3*).
- *Move Absolute* button. Opens the page in which the robot can be commanded to a, absolute position defining the motion parameters. Variables are mapped to the *CBT_MoveAbsolute* Function Block in the program example (*Figure 5.4.2*).
- *Move Circle* button. Opens the page in which the robot can be commanded for a circular movement defining the motion parameters. Variables are mapped to the *CBT_MoveCircular* Function Block in the program example (*Figure 5.4.4*).
- *PnP Sequence* Button. Opens the page in which the robot can be commanded to perform a PnP application. Each of the 6 motion segments can be configured. Variables are mapped to the *CBT_ProgramControl* Function Block in the program example (*Figure 5.4.7*).

6.3.2. pgConnect

This page is used as interface for *CBT_Connect* Function Block. Inputs can be configured and outputs are monitored. It is composed by 2 areas:

- *Connection Settings*, where Robot IP address, connection port and the timeout (before the FB triggers an error if connection cannot be done) must be defined. *Preset Values* button is used to set predefined values configured in the page script for input parameters.
- *Command Status*, region where the FB outputs are displayed to monitor FB status.

Connect button executes *CBT_Connect* Function Block (*Figure 5.4.1*) and *Main Menu* button returns to *Main* page.

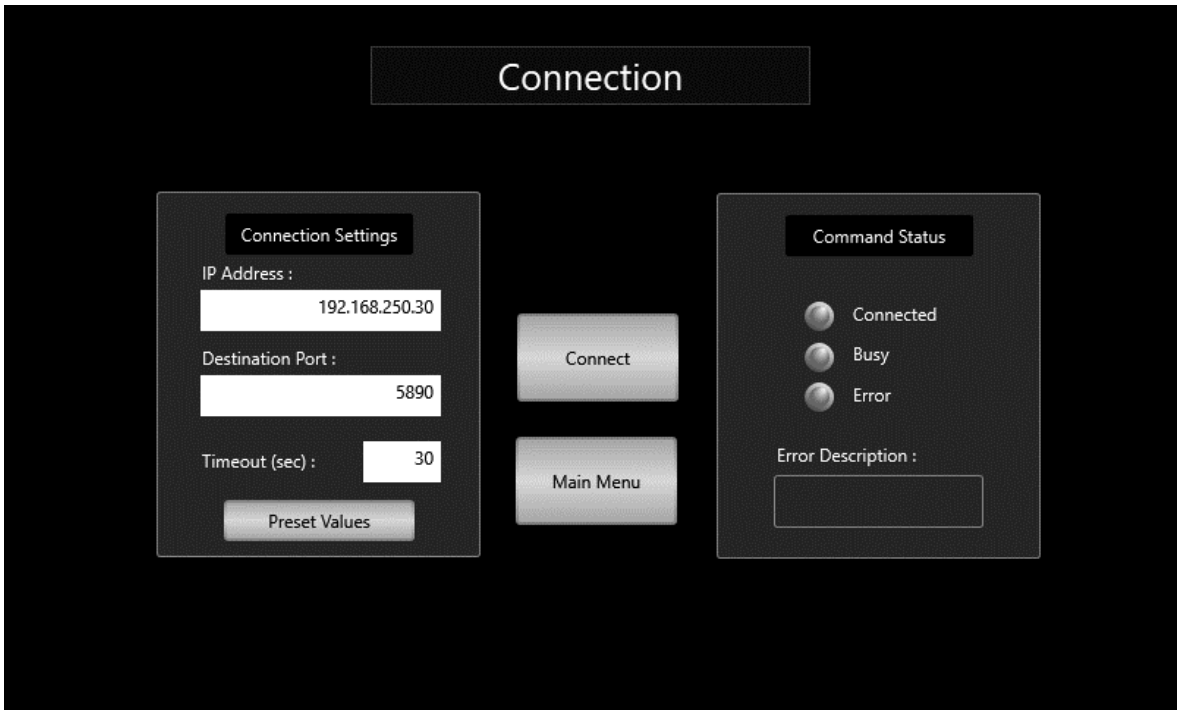

Figure 6.3.2 – pgConnect page of HMI (source: own).

6.3.3. pgChangeBase

This page is used as interface for *CBT_ChangeBase* Function Block. Inputs can be configured and outputs are monitored. It is composed by 2 areas:

- *Base Parameters,* where the base coordinate (mm, ◦) must be set. *Set Default Robot Base* button is used to set predefined values configured in the page script for input parameters. In this case, to reset the base coordinates.
- *Command Status,* region where the FB outputs are displayed to monitor FB status.

Execute button executes *CBT_ChangeBase* Function Block (*Figure 5.4.5*) and *Main Menu* button returns to *Main* page.

Figure 6.3.3 – pgChangeBase page of HMI (source: own).

6.3.4. pgChangeTCP

This page is used as interface for *CBT_ChangeTCP* Function Block. Inputs can be configured and outputs are monitored. It is composed by 2 areas:

- *TCP Parameters,* where the TCP offset (mm, ◦), Weight (kg), Inertia (kg·mm²) and Mass Center (mm, ◦) coordinates must be set. *Set Null Tool Offset* button is used to set predefined values configured in the page script for input parameters. In this case, null values for input parameters like no tool was selected.
- *Command Status,* region where the FB outputs are displayed to monitor FB status.

Execute button executes *CBT_ChangeTCP* Function Block (*Figure 5.4.6*) and *Main Menu* button returns to *Main* page.

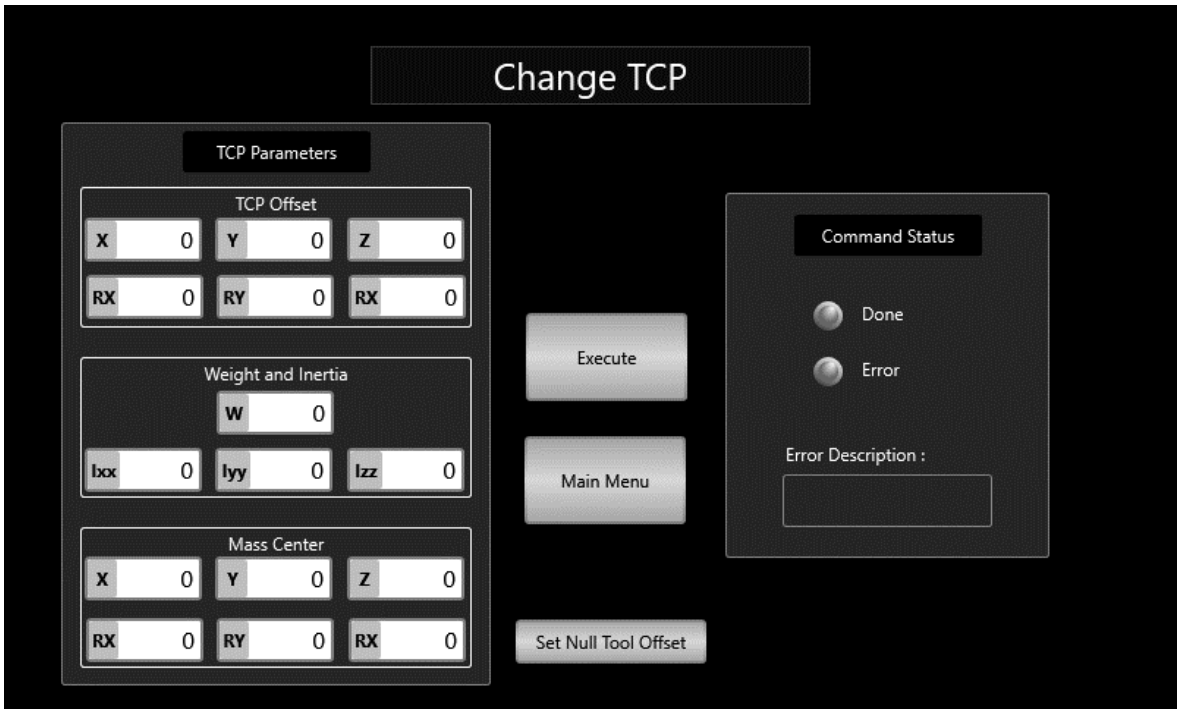

Figure 6.3.4 – pgChangeTCP page of HMI (source: own).

6.3.5. pgMoveRelative

This page is used as interface for *CBT_MoveRelative* Function Block. Inputs can be configured and outputs are monitored. It is composed by 2 areas:

- *Motion Parameters,* where the Target Position (mm, ◦), Motion Command (Line or PTP) and Motion Settings must be set. *Preset Values* button is used to set predefined values configured in the page script for input parameters. In this case, null values. *Enable Blending* and *Enable Precise Point* are also optional settings.
- *Command Status,* region where the FB outputs are displayed to monitor FB status.

Execute button executes *CBT_MoveRelative* Function Block (*Figure 5.4.3*) and *Main Menu* button returns to *Main* page.

Figure 6.3.5 – pgMoveRelative page of HMI (source: own).

6.3.6. pgMoveAbsolute

This page is used as interface for *CBT_MoveAbsolute* Function Block. Inputs can be configured and outputs are monitored. It is composed by 2 areas:

- *Motion Parameters,* where the Target Position (mm, ◦), Robot Pose (Righty/Lefty, Above/Below and Flip/Noflip), Motion Command (Line or PTP) and Motion Settings must be set. *Preset Values* button is used to set predefined values configured in the page script for input parameters. *Enable Blending* and *Enable Precise Point* are also optional settings.
- *Command Status,* region where the FB outputs are displayed to monitor FB status.

Execute button executes *CBT_MoveAbsolute* Function Block (*Figure 5.4.2*) and *Main Menu* button returns to *Main* page.

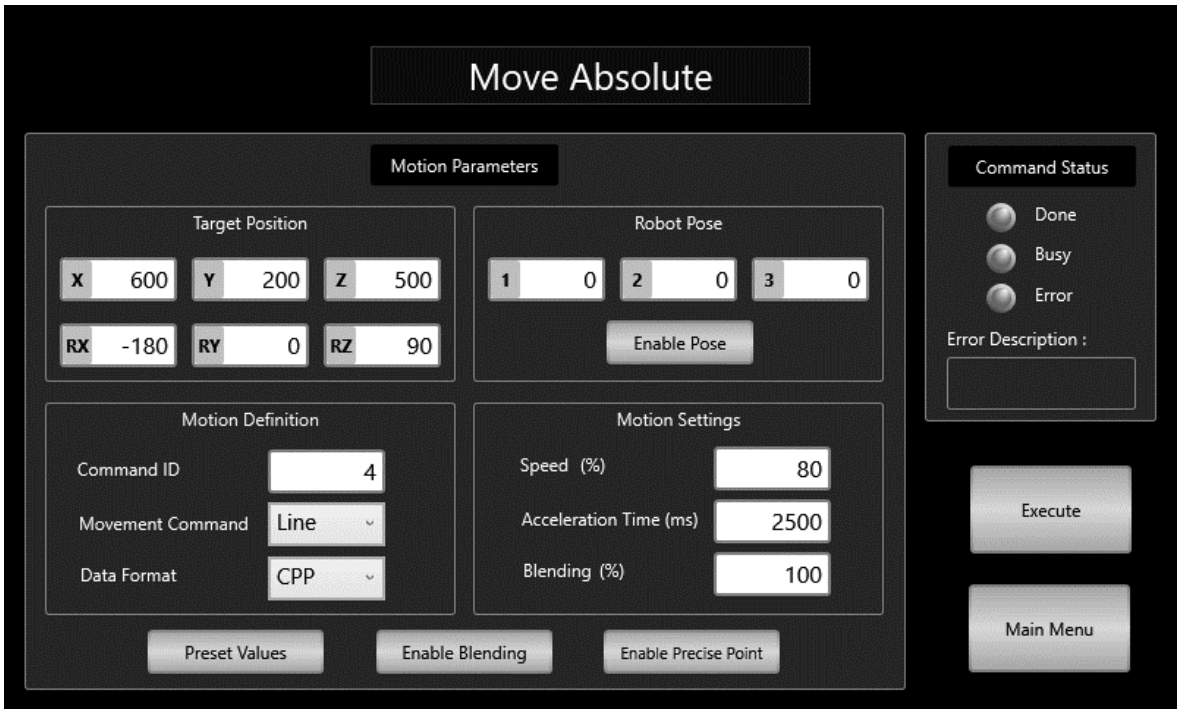

Figure 6.3.6 – pgMoveAbsolute page of HMI (source: own).

6.3.7. pgMoveCircle

This page is used as interface for *CBT_MoveCircle* Function Block. Inputs can be configured and outputs are monitored. It is composed by 2 areas:

- *Motion Parameters,* where the Arc Position and End Position (mm, ◦), Motion Command (Line or PTP) and Motion Settings must be set. *Preset Values* button is used to set predefined values configured in the page script for input parameters. *Enable Blending* and *Enable Precise Point* are also optional settings.
- *Command Status,* region where the FB outputs are displayed to monitor FB status.

Execute button executes *CBT_MoveCircle* Function Block (*Figure 5.4.4*) and *Main Menu* button returns to *Main* page.

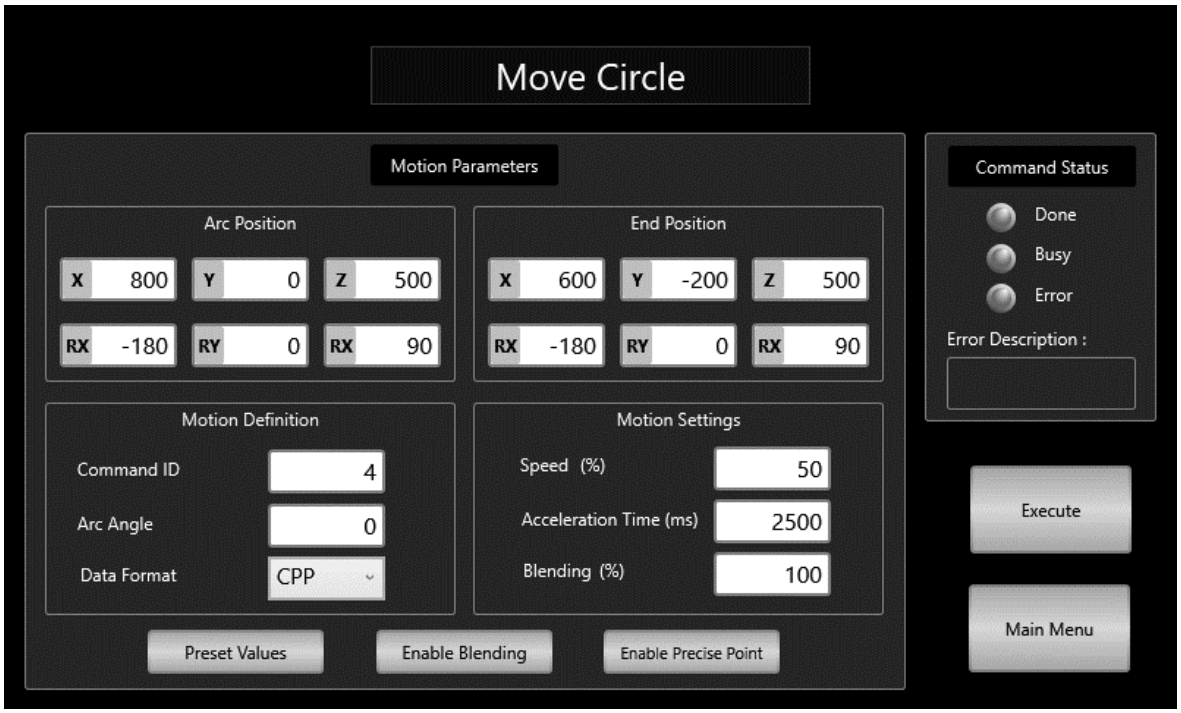

Figure 6.3.7 – pgMoveCircle page of HMI (source: own).

6.3.8. pgPnP_sequence

Page used as interface for Pick and Place application program in which each of the 6 motion segments are done by *CBT_MoveAbsolute* FB. It is composed by 3 areas:

- *Main* area where the trajectory is shown in 6 discrete movements which parameters can be modified by pressing *Edit* buttons.
- *Commands,* where *Preset Values* button is used to set predefined values configured in the page script for input parameters. In this case, coordinates, and motion strategy (speed, acceleration, PTP or line motion, etc.) for each one of the 6 movements. Additionally, this page includes *Pause* and *Resume* buttons to execute *CBT_ProgramControl* Function Block (*Figure 5.4.7*). If the program is paused during robot movement, the robot will complete the interrupted movement once the program execution is restored.
- *Command Status,* region where the FB outputs are displayed to monitor FB status.

Execute button executes *CBT_MoveAbsolute* FB and *Main Menu* button returns to *Main* page.

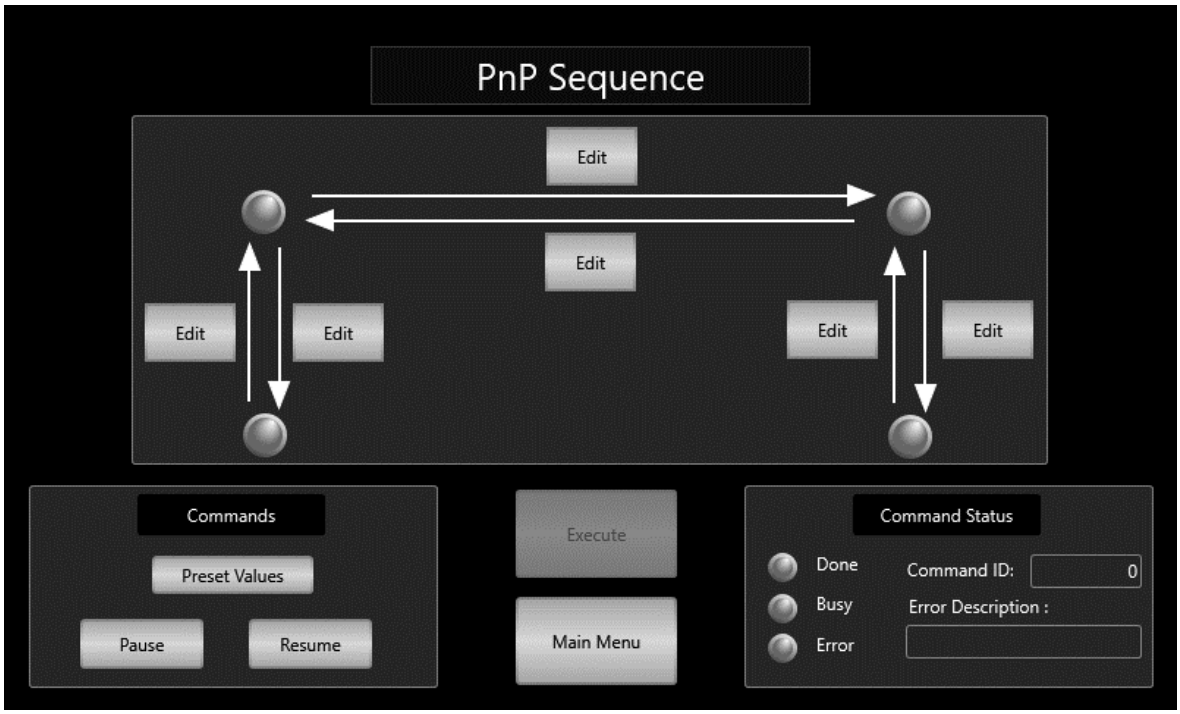

Figure 6.3.8 – pgPnP_Sequence main page of HMI (source: own).

Whether any *Edit* button is pressed, following window popups to provide to the usr the *Motion Parameters* of *CBT_MoveAbsolute*. Each one of the 6 movements has the same input parameters shown in the picture below. To select which movement the user wants to configure, just click on the *Edit* button close to the arrow representing the movement.

Figure 6.3.9 – pgPnP_Sequence settings page of HMI (source: own).

7. Environmental impact analysis

The product developed in this thesis is a set of Function Blocks, which usage would not directly affect to the environment. But the complete system will require electrical power thus the fact of producing that electricity would affect to the environment depending on which technology has been used by the company providing the service.

Conclusions

The first step of this project was defining the specifications of the product it was decided to create. Some of those specifications have been slightly modified during the development of the project due to the fact that some issue was found that forced to update some of those initial specifications.

The thesis is based on Function Blocks development, therefore the very first step was the search of regulations or guidelines to define the minimum requirements of a FB acceptance. Thus, a first FB sample was developed, tested, and debugged; achieving the expected appearance and behaviour. At that time, since the created FB sample was very robust in behaviour, it was decided to use that sample as template for all FBs in the library. Then, that template reduced the development time because its reuse.

Another topic was the way in which the FB identifies the robot to which the command is sent. This is done with the IP address of the robot. The idea was that the user introduces the IP address into the FB in a string variable, but there were possibilities of mistaken this. Therefore, it was required to find an intuitive and elegant solution, so it was conceived the in-out variable. It is common for all the FB and the IP address would be set only in the in/out variable definition, then just assigning this variable to the parameter, the robot is linked with the FB. As result, an intuitive and elegant way for the user to set the robot object.

Next point was the way in which the user introduces the input parameters in the FB. Since the message sent to the robot is a string variable, the first idea was to allow to the user to set the parameters as string format, what reduces the programming complexity, but this would imply a huge probability of wrong input parameters if the user does not know the proper syntax of the commands. Avoiding this undesired situation and considering as a must that the robot must receive the command message in the correct syntax, it was decided to define a datatype for each parameter. Reducing the chances of inputting wrong values and the possibility of having the robot rejecting commands because of wrong syntax. For those parameters which the value is string type, enumerated variables were created so the user just needs to select the item from a list and the FB will internally convert all input parameters in string variables and joining them in just one message. These all decisions helped to get robust, efficient, and elegant Function Blocks.

Regarding the capabilities of this Library in terms of number of robots that can be controlled, even though the development and testing have been done with just one robot connected to the communications network, the estimation is that 1 PLC would be able to control as many robots as the maximum number of TCP connections are allowed by the Omron controller used in the testing setup. Therefore, expectation is 32 cobots maximum.

In case of a second version would be release many improvements can be applied. Some of the new specifications could be:

- Include the missing functions available for cobot control from external devices.
- CommandID input parameter would be hidden for the user, and it would be managed internally by the FBs.
- In the CBT_MoveAbsolute Function Block, the RobotPose input parameter (*stCBT_RobotPose, table 5.3.2.7*) is integer datatype with only 2 possible values each one. The improvement is to define enumerated datatype to be clearer for the user:
    - 0 value would have “Righty” name.
    - 1 value would have “Lefty” name.
    - 2 value would have “Above” name.
    - 3 value would have “Below” name.
    - 4 value would have “NoFlip” name.
    - 5 value would have “Flip” name.

In general terms, it can be concluded that the Function Block Library created achieves the main target proposed at the beginning of the project: ***Provide to customers a solution to command Omron Cobots from external Omron controllers in order to make easier the integration with industrial machines.***

Budget

There are two types of expenses: Hardware and Engineering. Hardware considers the devices required for the development and for running an application where Omron cobot is controller from Omron PLC. Engineering considers the development and documentation of the Library and HMI.

The table below splits the overall cost in that two main groups.

HARDWARE

#	Reference	Description	Price
1	TM5-700	Omron TM-series Collaborative Robot (3 weeks loan)	1,406.94 €
2	NJ501-1300	Omron NJ-series Programmable Logic Controllers (PLC)	4,812.48 €
3	NA5-9W001S-V1	Omron NA-Series Touch screen HMI, 9 inch, 800x480 resolution	2,898.65 €
4	W4S1-05B	Omron Industrial Ethernet Switching Hub	320.46 €
5	S8VK-G01524	Omron Power Supply, 15 W, 24VDC	44.21 €
6	XS6W-6LSZH8SS100CM-G	Omron Ethernet cables 1 meter (4 units)	64.56 €
7	SYSMAC-SA401L-64	Omron Sysmac Studio programming software License	2,135.95 €
8	Tmflow	Omron Collaborative robot programming software	- €
			11,683.25 €
		IVA (21%)	2,453.48 €
		TOTAL HARDWARE	**14,136.73 €**

ENGINEERING

#	Reference	Description	Price
1	Dev-Prj	Development (150h x 28.95€)	4,342.50 €
2	Doc-Prj	Documentation (30h x 16.52€)	496.50 €
			4,839.00 €
		IVA (21%)	1,016.19 €
		TOTAL ENGINEERING	**5,855.19 €**

TOTAL PROJECT	**19,991.92 €**

Total cost estimation of the project is 19,991.92 €.

Bibliography

[1] PLCopen Organization. < https://plcopen.org/ >

[2] Omron Europe. *Product website* .< https://industrial.omron.eu/en/home >

[3] PLCopen, *Software Construction Guidelines. Creating PLCopen Compliant Libraries*. <https://plcopen.org/sites/default/files/downloads/creating_plcopen_compliant_function_block_libraries.pdf >

[4] International Electrotechnical Commission IEC Webstore. IEC 61131-3:2013. *Programmable controllers - Part 3: Programming languages*. <https://webstore.iec.ch/publication/4552 >

[5] AKSIT, Mehmet; TEKINERDOGAN, Bedir; BERGMANS, Lodewijk. *The six concerns for separation of concerns*. En 15th European Conference on Object-Oriented Programming, ECOOP 2001. 2001.

[6] Omron Europe. NJ_NX-series CPU Unit Built-in EtherNetIP Port User's Manual (W506-E1-24).

[7] Omron Europe. NJ_NX-series Instructions Reference Manual (W502-E1-32)

[8] Omron Europe. NJ-series CPU Unit Hardware User's Manual (W500-E1-27)

[9] Omron Europe. Sysmac Studio Version 1 Operation Manual (W504-E1-36)

[10] Omron Europe. Software Manual TMflow SW1.82 (I626-E-09)

[11] Omron Europe. TM Expression Editor and Listen Node Reference Guide (I848-E-03)

[12] Omron Europe. NA-series Connection (V119-E1-09)

[13] Omron Europe. NA-series hardware (V117-E1-10)

[14] Omron Europe. NA-series Software (V118-E1-18)

Annex A: Function Block scripts

A1. CBT_Connect

```
(* ************************************************************************************ *)
(*                              Robot StartUp FB                                        *)
(* ************************************************************************************ *)
(* ---------------------------------------------------------------------------
FB name:                                CBT_Connect
(* ---------------------------------------------------------------------------
FB version:             V01
Library:                CBT_Commands
SS version:             V1.44
Date:                   June 2021
Author:                 Ruben Sanchez Boli
Description:            Establishes TCP socket connection
(* ---------------------------------------------------------------------------- *)


// Detect rising flag on Execute input
R_TRIG_Execute(Clk:=Execute, Q=>flagExecute);

//Sequence initialization
IF flagExecute THEN
    IF FB_status <> 2 THEN
            //initates secuences
            FB_status:=1;
            FB_step:=10;

            //Save input at FB Execute to avoid input values modification during FB execution
            Local_PortIP        := 0;                   // Local TCP port number: Automatically assigned.
            AddressIP           := IPAddress;           // Cobot IP address
            Dest_PortIP         := IPPort;              // Cobot IP port
            ConnectionTimeOut   := Time_Out;            // Connection Timeout
    END_IF;
END_IF;

//Close connection
F_TRIG_Execute(Clk:=Execute, Q=>FalseExecute);
IF FalseExecute THEN
    FB_step:=100;
END_IF;


(* FB State Diagram control                                                         *)
(* -------------------------------------------------------------------------------- *)
//FBstatus_Step :  0 - Idle
//FBstatus_Step :  1 - Reset
//FBstatus_Step :  2 - Busy
//FBstatus_Step :  3 - Error
//FBstatus_Step :  4 - Error deactive
//FBstatus_Step :  5 - Done active
//FBstatus_Step :  6 - Done deactive


//Sequence
CASE FB_status OF

0: (* Idle         *)
            FB_status:=0;

1: (*Reset        *)
            ToError_Sts:=FALSE;
            ToDone_Sts:=FALSE;

            Done := FALSE;
            Busy := FALSE;
```

```
64                  Error := FALSE;
65                  ErrorID:=0;
66                  ErrorDescrip:=' ';
67                  FB_status:=2;
68
69  2: (*           Busy state *)
70                  Done := FALSE;
71                  Busy := TRUE;
72                  Error := FALSE;
73                  ErrorID:=0;
74                  ErrorDescrip:=' ';
75
76                  IF ToError_Sts THEN
77                          FB_status:=3;                   //Error flag
78                  END_IF;
79
80                  IF ToDone_Sts THEN
81                          FB_status:=5;                   //Done flag
82                  END_IF;
83
84  3: (*           Error state *)
85                  Done := FALSE;
86                  Busy := FALSE;
87                  Error := TRUE;
88                  ErrorID:=ToErrorID;
89                  ErrorDescrip:=ToErrorDescrip;
90
91                  IF NOT Execute THEN
92                          FB_status:= 4;                                  //Returns to idle
93                  END_IF;
94
95  4: (*           Error Deactive if not execute       *)
96                  Done := FALSE;
97                  Busy := FALSE;
98                  Error := FALSE;
99                  ErrorID:=ToErrorID;
100                 ErrorDescrip:=ToErrorDescrip;
101
102 5: (*           Done state *)
103                 Done := TRUE;
104                 Busy := FALSE;
105                 Error := FALSE;
106                 ErrorID:=0;
107                 ErrorDescrip:=' ';
108
109                 IF NOT Execute THEN
110                         FB_status:= 6;                                  //Returns to idle
111                 END_IF;
112
113 6: (*                   Done deactive if not Execute                *)
114                 Done := FALSE;
115                 Busy := FALSE;
116                 Error := FALSE;
117                 ErrorID:=0;
118                 ErrorDescrip:=' ';
119
120 END_CASE ;
121
122
123 (* FB Algorithm                                                    *)
124 (* ---------------------------------------------------------------- *)
125
126 IF Busy THEN
127
128     CASE FB_step OF
129     0: (*       Idle                    *)
130                 FB_step:=0;
131
132     10: (*      Initialization          *)
133                 //InternalStep variables
134                 TCP_Connect_Exe                 :=FALSE;
```

```
135                     TCP_Clear_Buffer_Exe        :=FALSE;
136                     Get_TCP_Status_Exe          :=FALSE;
137                     TCP_Send_Exe                                :=FALSE;
138                     TCP_Rcv_Exe                                 :=FALSE;
139                     TCP_Close_Exe                               :=FALSE;
140                     TON_TimeOut_Exe             :=FALSE;
141                     ToErrorID                                           :=0;
142                     ToErrorDescrip                              :=' ';
143
144                     //FB Outputs
145                     CBT.WaitingReturn:=FALSE;
146                     CBT.Connected:=FALSE;
147                     Connected:=CBT.Connected;
148                     LastConnTime        := SecToDt(0);
149                     LastLostTime        := SecToDt(0);
150                     FirstConnection:=FALSE;
151
152                     FB_step:=15;
153
154     15:(*       Check Timeout value             *)
155
156                     IF (ConnectionTimeOut<10) OR (ConnectionTimeOut>300) THEN
157                             ToErrorID:=16#15;
158                             ToErrorDescrip:='Timeout range: 10~300 seconds';
159                             TCP_Connect_Exe:=FALSE;
160                             ToError_Sts :=TRUE;
161                     else
162                             FB_step:=20;
163                     END_IF;
164
165
166
167     20:(*       Request a connection            *)
168                     TCP_Connect_Exe := TRUE;
169                     TON_TimeOut_Exe:=TRUE;
170
171                     IF TCP_Connect.Done THEN
172                             TCP_Connect_Exe:=FALSE;
173                             TON_TimeOut_Exe:=FALSE;
174                             FB_step:=30;
175                     END_IF;
176
177                     IF TCP_Connect.Error THEN
178                             ToErrorID:=16#20;
179                             ToErrorDescrip:='Connection Error';
180                             TCP_Connect_Exe:=FALSE;
181                             ToError_Sts :=TRUE;
182                     END_IF;
183
184                     IF TimeOut THEN
185                             ToErrorID:=16#21;
186                             ToErrorDescrip:='TimeOut Error';
187                             TCP_Connect_Exe:=FALSE;
188                             TON_TimeOut_Exe:=FALSE;
189                             ToError_Sts :=TRUE;
190                     END_IF;
191
192     30: (*      Clear receive buffer            *)
193                     TCP_Clear_Buffer_Exe:=TRUE;
194
195                     IF TCP_Clear_Buffer.Done THEN
196                             TCP_Clear_Buffer_Exe:=FALSE;
197                             FB_step:=40;
198                     END_IF;
199
200                     IF TCP_Clear_Buffer.Error THEN
201                             ToErrorID:=16#30;
202                             ToErrorDescrip:='Clear Buffer Error';
203                             TCP_Clear_Buffer_Exe:=FALSE;
204                             ToError_Sts :=TRUE;
205                     END_IF;
```

```
206
207
208     40: (*      Request reading status              *)
209                 Get_TCP_Status_Exe:=TRUE;
210
211                 IF Get_TCP_Status.Done THEN
212                     Get_TCP_Status_Exe:=FALSE;
213                     FB_step:=50;
214                 END_IF;
215
216                 IF Get_TCP_Status.Error THEN
217                     ToErrorID:=16#40;
218                     ToErrorDescrip:='Get TCP Status Error';
219                     Get_TCP_Status_Exe:=FALSE;
220                     ToError_Sts := TRUE;
221                 END_IF;
222
223
224     50: (*      Request sending data                *)
225                 // Converts command to byte array
226
227                 str_CheckListen:='$$TMSTA,2,00,*41$R$L';
228                 // $$TMSTA          --> Communication package acquiring status // Sysmac requires $$, the 1st is
    sysmac systax to recognize the 2nd $ as character
229                 // 2             --> Indicates the length of 00 is 2 bytes
230                 // 00            --> Indicates if cobot is in external script control mode or not (is Cobot in Listen node or
    not)
231                 // 41            --> Checksum
232                 //$R             --> $R in sysmac syntax is \R in ASCII (carriage)
233                 //$L             --> $L in sysmac syntax is \L in ASCII (enter)
234
235                 TCP_Send_Size:=LEN(str_CheckListen);
236                 ToAryByte(In:=str_CheckListen, Order:=_eBYTE_ORDER#_LOW_HIGH,
    AryOut:=TCP_Send_Data[0]);
237
238
239                 TCP_Send_Exe        :=TRUE;
240
241                 IF TCP_Send.Done THEN
242                     TCP_Send_Exe:=FALSE;
243                     FB_step:=60;
244                 END_IF;
245
246                 IF TCP_Send.Error THEN
247                     ToErrorID:=16#50;
248                     ToErrorDescrip:='TCP Send Error';
249                     TCP_Send_Exe:=FALSE;
250                     ToError_Sts := TRUE;
251                 END_IF;
252
253     60: (*      Request receiving data                  *)
254
255                 TCP_Rcv_TimeOut:=0;         //0: No timeouts
256                 TCP_Rcv_Size:=256;          //Set number of bytes to read from the receive buffer
257                 StringOfReceivedData:='';   //Clear the variable where Receive data array is compiled
258
259                 TCP_Rcv_Exe         :=TRUE;
260
261                 IF TCP_Rcv.Done THEN
262                     StringOfReceivedData:=AryToString(TCP_Rcv_Data[0],TCP_Rcv_RcvSize);
                    //Converts a maximum of 1985 BYTE array to a text string (starting from index [0])
263                     TCP_Rcv_Exe:=FALSE;
264                     FB_step:=70;
265                 END_IF;
266
267                 IF TCP_Rcv.Error THEN
268                     ToErrorID:=16#60;
269                     ToErrorDescrip:='TCP Receive Error';
270                     TCP_Rcv_Exe:=FALSE;
271                     ToError_Sts := TRUE;
272                 END_IF;
```

```
273
274
275     70: (*      Check received data                 *)
276
277                         IF FIND(StringOfReceivedData,'true') <> 0 THEN
278                                 FB_step:=80;
279                         ELSIF FIND(StringOfReceivedData,'false') <> 0 THEN
280                                 ToErrorID:=16#70;
281                                 ToErrorDescrip:='Device is not a Cobot';
282                                 ToError_Sts := TRUE;
283                         END_IF;
284
285
286     80: (*      Continuously checking status                *)
287
288                         Get_TCP_Status_Exe:=TRUE;
289
290                         IF Get_TCP_Status.Done THEN
291                                 CBT.Connected:=TRUE;
292                                 Connected:=CBT.Connected;
293                                 Get_TCP_Status_Exe:=FALSE;
294                                 FB_step:=90;
295                         END_IF;
296
297                         IF Get_TCP_Status.Error THEN
298                                 ToErrorID:=16#80;
299                                 ToErrorDescrip:='Get TCP Status Error';
300                                 Get_TCP_Status_Exe:=FALSE;
301                                 ToError_Sts := TRUE;
302                         END_IF;
303
304
305     90: (*      Check status each time                  *)
306
307                         IF NOT(FirstConnection) THEN //Updates output just once
308                                 LastConnTime:=GetTime();
309                                 FirstConnection:=TRUE;
310                         END_IF;
311
312                         Timer_1_Enable := TRUE ;
313
314                         IF Timer_1_Done THEN
315                                 Timer_1_Enable:=FALSE;
316                                 FB_step:=FB_step-10;                    //Continuously checking status
317                         END_IF;
318
319                         IF SocketStatus =_CLOSE_WAIT THEN
320                                 LastLostTime:=GetTime();
321                                 ToErrorID:=16#90;
322                                 ToErrorDescrip := 'Socket disconnected';
323                                 CBT.WaitingReturn:=FALSE;
324                                 CBT.Connected:=FALSE;
325                                 Connected:=CBT.Connected;
326                                 Get_TCP_Status_Exe:=FALSE;
327                                 ToError_Sts := TRUE;
328                         END_IF;
329
330
331     100: (*     Request closing                 *)
332
333                         CBT.WaitingReturn:=FALSE;
334                         CBT.Connected:=FALSE;
335                         Connected:=CBT.Connected;
336                         TCP_Close_Exe:=TRUE;
337                         LastLostTime:=GetTime();
338
339                         IF TCP_Close.Done THEN
340                                 TCP_Close_Exe:=FALSE;
341                                 FB_step:=110;
342                         END_IF;
343
```

```
344                             IF TCP_Close.Error THEN
345                                     ToErrorID:=16#100;
346                                     ToErrorDescrip:='TCP Close Error';
347                                     TCP_Close_Exe:=FALSE;
348                                     ToError_Sts := TRUE;
349                             END_IF;
350
351
352         110: (*     End Execution                   *)
353                     ToDone_Sts:=TRUE;
354
355         END_CASE ;
356
357     END_IF;
358
359
360     (* Function Bolcks                                          *)
361     (* ---------------------------------------------------------- *)
362
363     TCP_Connect(
364         Execute:=TCP_Connect_Exe,
365         SrcTcpPort:=Local_PortIP,
366         DstAdr:=AddressIP,
367         DstTcpPort:=Dest_PortIP,
368         //Done=>, Busy=>, Error=>, ErrorID=>,
369         Socket=>CBT.Socket);
370
371     TCP_Clear_Buffer(
372         Execute:=TCP_Clear_Buffer_Exe,
373         Socket:=CBT.Socket
374         //Done=>, Busy=>, Error=>, ErrorID=>
375         );
376
377     Get_TCP_Status(
378         Execute:=Get_TCP_Status_Exe,
379         Socket:=CBT.Socket,
380         //Done=>, Busy=>, Error=>, ErrorID=>,
381         TcpStatus=>SocketStatus
382         //DatRcvFlag=>
383         );
384
385     TCP_Send(
386         Execute:=TCP_Send_Exe,
387         Socket:=CBT.Socket,
388         SendDat:=TCP_Send_Data[0],
389         Size:=TCP_Send_Size
390         //Done=>, Busy=>, Error=>, ErrorID=>
391         );
392
393     TCP_Rcv(
394         Execute:=TCP_Rcv_Exe,
395         Socket:=CBT.Socket,
396         TimeOut:=TCP_Rcv_TimeOut,
397         Size:=TCP_Rcv_Size,
398         RcvDat:=TCP_Rcv_Data[0],
399         //Done=>, Busy=>, Error=>, ErrorID=>,
400         RcvSize=>TCP_Rcv_RcvSize);
401
402     TCP_Close(
403         Execute:=TCP_Close_Exe,
404         Socket:=CBT.Socket
405         //Done=>, Busy=>, Error=>, ErrorID=>
406         );
407
408     TON_TimeOut(In:=TON_TimeOut_Exe, PT:=NanoSecToTime(ConnectionTimeOut*1000000000), Q=>TimeOut);
409     Timer_1(In:=Timer_1_Enable, PT:=T#1000ms, Q=>Timer_1_Done);
```

A2. CBT_MoveAbsolute

```
(* ************************************************************************** *)
(*                                Robot StartUp FB                            *)
(* ************************************************************************** *)
(* -------------------------------------------------------------------------- *)
FB name:                        CBT_MoveAbsolute
(* -------------------------------------------------------------------------- *)
FB version:             V01
Library:                CBT_Commands
SS version:             V1.44
Date:                   June 2021
Author:                 Ruben Sanchez Boli
Description:            Sends motion commands to absolute positions
                        - PTP() -> Syntax 4 in expression editor
                        - Line() -> Syntax 2 in expression editor
(* -------------------------------------------------------------------------- *)


// Detect rising flag on Execute input
R_TRIG_Execute(Clk:=Execute, Q=>flagExecute);

//Sequence initialization
IF flagExecute THEN
    IF FB_status <> 2 THEN              //FBstatus_Step different than 2 - Busy (avoids re-execution)
            iParameters:=Parameters;    //Copy internal variable
            FB_status:=1;               //Initates State Diagram control
            FB_step:=10;                //Initates algorithm sequence
    END_IF;
END_IF;

//If robot is not conneted when FB is executed or robot suddenly disconnects (e.g. EStop is triggered during FB execution)
IF Execute AND NOT(CBT.Connected) THEN
    ToError_Sts := TRUE;
    ToErrorID:=16#99;
    ToErrorDescrip:='Robot not connected';
END_IF;

(* FB State Diagram control                                                   *)
(* -------------------------------------------------------------------------- *)
//FBstatus_Step :  0 - Idle
//FBstatus_Step :  1 - Reset
//FBstatus_Step :  2 - Busy
//FBstatus_Step :  3 - Error
//FBstatus_Step :  4 - Error deactive
//FBstatus_Step :  5 - Done active
//FBstatus_Step :  6 - Done deactive


//Sequence
CASE FB_status OF

0: (*                   Idle                        *)
        FB_status:=0;

1: (*                   Reset               *)
        ToError_Sts:=FALSE;
        ToDone_Sts:=FALSE;

        Done := FALSE;
        Busy := FALSE;
        Error := FALSE;
        ErrorID:=0;
        ErrorDescrip:=' ';
        FB_status:=2;

2: (*   Busy state *)
        Done := FALSE;
        Busy := TRUE;
        Error := FALSE;
```

```
69                  ErrorID:=0;
70                  ErrorDescrip:=' ';
71
72                  IF ToError_Sts THEN
73                          FB_status:=3;                   //"Error" flag
74                  END_IF;
75
76                  IF ToDone_Sts THEN
77                          FB_status:=5;                   //"Done" flag
78                  END_IF;
79
80
81  3: (*           Error state *)
82                  Done := FALSE;
83                  Busy := FALSE;
84                  Error := TRUE;
85                  ErrorID:=ToErrorID;
86                  ErrorDescrip:=ToErrorDescrip;
87
88                  IF NOT Execute THEN
89                          FB_status:= 4;                  //Returns to idle
90                  END_IF;
91
92  4: (*           Error Deactive if not execute       *)
93                  Done := FALSE;
94                  Busy := FALSE;
95                  Error := FALSE;
96                  ErrorID:=ToErrorID;
97                  ErrorDescrip:=ToErrorDescrip;
98
99
100 5: (*           Done state *)
101                 Done := TRUE;
102                 Busy := FALSE;
103                 Error := FALSE;
104                 ErrorID:=0;
105                 ErrorDescrip:=' ';
106
107                 IF NOT Execute THEN
108                         FB_status:= 6;                  //Returns to idle
109                 END_IF;
110
111 6: (*           Done deactive if not Execute                *)
112                 Done := FALSE;
113                 Busy := FALSE;
114                 Error := FALSE;
115                 ErrorID:=0;
116                 ErrorDescrip:=' ';
117
118 END_CASE ;
119
120
121
122 (* FB Algorithm                                             *)
123 (* ---------------------------------------------------------- *)
124
125 IF Busy THEN                        //FB State Diagram control      in "Busy" state
126
127     CASE FB_step OF
128     0: (*       Idle                    *)
129                 FB_step:=0;
130
131     10: (*      Initialization                  *)
132
133                 TCP_Clear_Buffer_Exe        :=FALSE;
134                 TCP_Send_Exe                                :=FALSE;
135                 TCP_Rcv_Exe                                 :=FALSE;
136                 ToErrorID                                           :=0;
137                 ToErrorDescrip                              :=' ';
138                 CmdID_Ack                                           :=Parameters.CommandID;
139
```

```
140                         IF CBT.Connected and NOT(CBT.WaitingReturn) THEN                    //If robot is connected
    and "released" (not occupied by other FB)
141                                 FB_step:=20;
142                         ELSIF NOT(CBT.Connected) THEN
                                    //Robot not connected
143                                 ToError_Sts := TRUE;
144                                 ToErrorID:=16#10;
145                                 ToErrorDescrip:='Robot not connected';
146                         ELSIF (CBT.WaitingReturn) THEN
                                            //Robot is "occupied" by other FB
147                                 ToError_Sts := TRUE;
148                                 ToErrorID:=16#11;
149                                 ToErrorDescrip:='Other FB is under execution';
150                         END_IF;
151
152
153     20: (*      Clear receive buffer            *)
154                         TCP_Clear_Buffer_Exe:=TRUE;
155
156                         IF TCP_Clear_Buffer.Done THEN
157                                 TCP_Clear_Buffer_Exe:=FALSE;
158                                 FB_step:=25;
159                         END_IF;
160
161                         IF TCP_Clear_Buffer.Error THEN
162                                 TCP_Clear_Buffer_Exe:=FALSE;
163                                 ToError_Sts := TRUE;
164                                 ToErrorID:=16#20;
165                                 ToErrorDescrip:='Clear buffer error';
166                         END_IF;
167
168
169     25: (*      Do not exceed the seetable range *)
170
171                         IF iParameters.MovementCommand=\\CBT_Commands_Lib\eCBT_MovCmd_Movement#Line
    THEN
172                                 IF iParameters.Speed > 4500 THEN
173                                         ToError_Sts := TRUE;
174                                         ToErrorID:=16#25;
175                                         ToErrorDescrip:='Speed range in Line: 0~4500 mm/s';
176                                 END_IF;
177                         END_IF;
178
179                         IF iParameters.MovementCommand=\\CBT_Commands_Lib\eCBT_MovCmd_Movement#PTP
    THEN
180                                 IF iParameters.Speed > 100 THEN
181                                         ToError_Sts := TRUE;
182                                         ToErrorID:=16#26;
183                                         ToErrorDescrip:='Speed range in PTP: 0~100 %';
184                                 END_IF;
185                         END_IF;
186
187                         IF (iParameters.AccelTime < 150) OR (iParameters.AccelTime > 9999) THEN
188                                 ToError_Sts := TRUE;
189                                 ToErrorID:=16#27;
190                                 ToErrorDescrip:='Acceleration Time range is: 150~9999 ms';
191                         END_IF;
192
193                         IF (iParameters.CommandID < 2) OR (iParameters.CommandID > 9) THEN
194                                 ToError_Sts := TRUE;
195                                 ToErrorID:=16#28;
196                                 ToErrorDescrip:='Command ID range is: 2~9';
197                         END_IF;
198
199                         IF iParameters.MovementCommand=\\CBT_Commands_Lib\eCBT_MovCmd_Movement#Line
    THEN
200                                 IF iParameters.RobotPoseEnable = true THEN
201                                         ToError_Sts := TRUE;
202                                         ToErrorID:=16#29;
203                                         ToErrorDescrip:='Robot Pose must be disable for Line command';
```

```
204                             END_IF;
205                     END_IF;
206
207                     FB_step:=30;
208
209
210     30: (*      Do not exceed the seetable range *)
211
212                     IF iParameters.RobotPoseEnable THEN
213                             IF (iParameters.RobotPose.Pose1 < 0) OR (iParameters.RobotPose.Pose1 > 1) THEN
214                                     ToError_Sts := TRUE;
215                                     ToErrorID:=16#30;
216                                     ToErrorDescrip:='Pose1 range is 0~1';
217                             END_IF;
218                     END_IF;
219
220                     IF iParameters.RobotPoseEnable THEN
221                             IF (iParameters.RobotPose.Pose2 < 2) OR (iParameters.RobotPose.Pose2 > 3) THEN
222                                     ToError_Sts := TRUE;
223                                     ToErrorID:=16#31;
224                                     ToErrorDescrip:='Pose2 range is 2~3';
225                             END_IF;
226                     END_IF;
227
228                     IF iParameters.RobotPoseEnable THEN
229                             IF (iParameters.RobotPose.Pose3 < 4) OR (iParameters.RobotPose.Pose3 > 5) THEN
230                                     ToError_Sts := TRUE;
231                                     ToErrorID:=16#32;
232                                     ToErrorDescrip:='Pose3 range is 4~5';
233                             END_IF;
234                     END_IF;
235
236                     FB_step:=35;
237
238     35: (*  iParameters conversion to string *)
239
240                     str_CommandID:=UINT_TO_STRING(iParameters.CommandID);
241
242                     // iParameters.MovementCommand
243                     IF iParameters.MovementCommand=\\CBT_Commands_Lib\eCBT_MovCmd_Movement#Line
    THEN
244                             str_MovementCommand:='Line';
245                     END_IF;
246
247                     IF iParameters.MovementCommand=\\CBT_Commands_Lib\eCBT_MovCmd_Movement#PTP
    THEN
248                             str_MovementCommand:='PTP';
249                     END_IF;
250
251                     //iParameters.DataFormat
252                     IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdAbs_DataFormat#CAP_Abs
    THEN
253                             str_DataFormat:='CAP';
254                     END_IF;
255
256                     IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdAbs_DataFormat#CAR_Abs
    THEN
257                             str_DataFormat:='CAR';
258                     END_IF;
259
260                     IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdAbs_DataFormat#CPP_Abs
    THEN
261                             str_DataFormat:='CPP';
262                     END_IF;
263
264                     IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdAbs_DataFormat#CPR_Abs
    THEN
265                             str_DataFormat:='CPR';
266                     END_IF;
267
```

```
268                             IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdAbs_DataFormat#JPP_Abs
    THEN
269                                     str_DataFormat:='JPP';
270                             END_IF;
271
272                             //Conversion of REAL variables to a text string with the specified format
273                             str_TP_X := RealToFormatString(In:=iParameters.TargetPosition.X, Exponent:=FALSE,
    Sign:=TRUE, MinLen:=1, DecPlace:=0);
274                             str_TP_Y := RealToFormatString(In:=iParameters.TargetPosition.Y, Exponent:=FALSE,
    Sign:=TRUE, MinLen:=1, DecPlace:=0);
275                             str_TP_Z := RealToFormatString(In:=iParameters.TargetPosition.Z, Exponent:=FALSE, Sign:=TRUE,
    MinLen:=1, DecPlace:=0);
276                             str_TP_RX := RealToFormatString(In:=iParameters.TargetPosition.RX, Exponent:=FALSE,
    Sign:=TRUE, MinLen:=1, DecPlace:=0);
277                             str_TP_RY := RealToFormatString(In:=iParameters.TargetPosition.RY, Exponent:=FALSE,
    Sign:=TRUE, MinLen:=1, DecPlace:=0);
278                             str_TP_RZ := RealToFormatString(In:=iParameters.TargetPosition.RZ, Exponent:=FALSE,
    Sign:=TRUE, MinLen:=1, DecPlace:=0);
279                             str_TP1:=CONCAT(str_TP_X,',',str_TP_Y,',',str_TP_Z);
280                             str_TP2:=CONCAT(str_TP_RX,',',str_TP_RY,',',str_TP_RZ);
281                             str_TargetPosition:=CONCAT(str_TP1,',',str_TP2);
282
283                             //Conversion of integer to text string
284                             str_Speed:=UINT_TO_STRING(iParameters.Speed);
285                             str_AccelTime:=UINT_TO_STRING(iParameters.AccelTime);
286
287                             IF iParameters.BlendingEnable THEN
288                                     str_BlendingValue:=UINT_TO_STRING(iParameters.BlendingValue);
289                             ELSE
290                                     str_BlendingValue:='0';
291                             END_IF;
292
293                             //Conversion of bool to text string
294                             IF iParameters.PrecisePositioning THEN
295                                     str_PrecisePositioning:='true';
296                             ELSE
297                                     str_PrecisePositioning:='false';
298                             END_IF;
299
300                             //iParameters.RobotPose
301                             str_RobotPose:=CONCAT(      UINT_TO_STRING(iParameters.RobotPose.Pose1),',',
302
                    UINT_TO_STRING(iParameters.RobotPose.Pose2),',',
303
                    UINT_TO_STRING(iParameters.RobotPose.Pose3));
304
305
306                             //Verify if dataformat input is valid for MovementCommand selection
307                             IF (str_MovementCommand='PTP') THEN
308                                     IF (str_DataFormat='JPP') OR (str_DataFormat='CPP') THEN
309                                             FB_step:=70;
310                                     ELSE
311                                             ToError_Sts := TRUE;
312                                             ToErrorID:=16#35;
313                                             ToErrorDescrip:='DataFormat invalid for PTP command';
314                                     END_IF;
315                             END_IF;
316
317                             IF (str_MovementCommand='Line') THEN
318                                     IF (str_DataFormat='CPP') OR
319                                             (str_DataFormat='CPR') OR
320                                             (str_DataFormat='CAP') OR
321                                             (str_DataFormat='CAR') THEN
322                                             FB_step:=70;
323                                     ELSE
324                                             ToError_Sts := TRUE;
325                                             ToErrorID:=16#36;
326                                             ToErrorDescrip:='DataFormat invalid for Line command';
327                                     END_IF;
328                             END_IF;
329
```

```
330                 FB_step:=40;
331
332         40: (*  Frame buildingfor CheckSum calculation *)
333
334                         Header:='TMSCT';                //Header required by robot controller to receive external scripts
335
336                         //Creates the script command with all the input parameters
337                         IF NOT(iParameters.RobotPoseEnable) then
338                                 Script_Command:=CONCAT(     CONCAT(str_MovementCommand,'(',str_DataFormat,','),
339
                                    CONCAT(str_TargetPosition,',',str_Speed,','),
340
                                    CONCAT(str_AccelTime,',',str_BlendingValue,','),
341
                                    CONCAT(str_PrecisePositioning,')') );
342                         ELSE
343                                 Script_Command:=CONCAT(     CONCAT(str_MovementCommand,'(',str_DataFormat,','),
344
                                    CONCAT(str_TargetPosition,',',str_Speed,','),
345
                                    CONCAT(str_AccelTime,',',str_BlendingValue,','),
346
                                    CONCAT(str_PrecisePositioning,',',str_RobotPose,')') );
347                         END_IF;
348
349
350                         //When no blending motion, acknowledgement with QueueTag()
351                         IF NOT(iParameters.BlendingEnable) THEN
352                                 Script_Command:=CONCAT(Script_Command,'$R$LQueueTag(',str_CommandID,',0)');
353                         END_IF;
354
355                         //Exits Listen Node in TMflow with ScriptExit()
356                         IF iParameters.ExitNode THEN
357                                 Script_Command:=CONCAT(Script_Command,'$R$L','ScriptExit()');
358                         END_IF;
359
360                         // CheckSum calculation
361                         str_Length:=INT_TO_STRING(LEN(CONCAT(str_CommandID,',',Script_Command)));

                    //Length for DATA command
362                         str_Checksum_Calc :=
    CONCAT(CONCAT(Header,',',str_Length,',',str_CommandID),CONCAT(',',Script_Command),',');                 //Checksum
    includes: Header, Length for DATA command, str_CommandID and Script_Command
363
364
        Checksum_Length:=ToAryByte(str_Checksum_Calc,_eBYTE_ORDER#_LOW_HIGH,Send_Checksum[0]);
365
366                         IF Checksum_Length>0 THEN
367                                 FB_step:=50;
368                         ELSE
369                                 ToError_Sts := TRUE;
370                                 ToErrorID:=16#40;
371                                 ToErrorDescrip:='Checksum length not valid';
372                         END_IF;
373
374
375
376         50: (*  Frame buildingto send command to TMflow *)
377
378                         //CheckSum calculation by XOR operation
379                         FOR i:=1 TO LEN(str_Checksum_Calc) BY 1 DO
380                                 Send_Checksum[0]:=Send_Checksum[0] XOR Send_Checksum[i];
381                         END_FOR;
382
383                         //Converts checksum byte value to text string
384                         str_Checksum:=BYTE_TO_STRING(Send_Checksum[0]);
385                         str_SendFrame :=
    CONCAT(CONCAT('$$',Header,',',str_Length,','),CONCAT(str_CommandID,',',Script_Command),',*',str_Checksum,'$R$L');
386
387                         //Finds the number of characters in the string to be sent in the frame
388                         Length:=LEN(str_SendFrame);
```

```
389                         Long:=ToAryByte(str_SendFrame, _eBYTE_ORDER#_LOW_HIGH,TCP_Send_Data[0]);

390
391                         IF Long>0 THEN
392                             FB_step:=60;
393                         ELSE
394                             ToError_Sts := TRUE;
395                             ToErrorID:=16#50;
396                             ToErrorDescrip:='Final frame building is error end';
397                         END_IF;
398
399
400     60: (*      Send command                    *)
401
402                         //Finds the number of characters in the string to be sent in the frame
403                         TCP_Send_Size:=LEN(str_SendFrame);
404                         ToAryByte(In:=str_SendFrame, Order:=_eBYTE_ORDER#_LOW_HIGH,
    AryOut:=TCP_Send_Data[0]);
405
406                         TCP_Send_Exe        :=TRUE;
407
408                         IF TCP_Send.Done THEN
409                             CBT.WaitingReturn:=TRUE;            //Flag to set the robot in "busy" state to avoid
    other FB to be executed
410                             TCP_Send_Exe:=FALSE;
411                             FB_step:=70;
412                         END_IF;
413
414                         IF TCP_Send.Error THEN
415                             TCP_Send_Exe:=FALSE;
416                             ToError_Sts := TRUE;
417                             ToErrorID:=16#60;
418                             ToErrorDescrip:='TCP send error';
419                         END_IF;
420
421
422     70: (*      Request receiving data              *)
423
424                         TCP_Rcv_TimeOut:=0;                                 //0: No timeouts
425                         TCP_Rcv_Size:=256;                                  //Set number of bytes to read from
    the receive buffer
426                         StringOfReceivedData:='';           //Clear the variable where Receive data array is compiled
427
428                         TCP_Rcv_Exe :=TRUE;
429
430                         IF TCP_Rcv.Done THEN
431                             StringOfReceivedData:=AryToString(TCP_Rcv_Data[0],TCP_Rcv_Size);
                                                                //Converts a maximum of 1985 BYTE array to
    a text string (starting from index [0])
432                             TCP_Rcv_Exe:=FALSE;
433                             FB_step:=80;
434                         END_IF;
435
436                         IF TCP_Rcv.Error THEN
437                             TCP_Rcv_Exe:=FALSE;
438                             ToError_Sts := TRUE;
439                             ToErrorID:=16#70;
440                             ToErrorDescrip:='TCP receive error';
441                         END_IF;
442
443
444     80: (*      Check acknowledgement Command accepted          *)
445
446                 IF FIND(StringOfReceivedData,'TMSTA') <> 0 THEN             //Message no vaild
447                     FB_step:=70;
448                 END_IF;
449
450                 IF FIND(StringOfReceivedData,'TMSCT') <> 0 THEN
451                     IF FIND(StringOfReceivedData,'OK') <> 0 THEN                    //Command
    accepted
452                         IF NOT(iParameters.BlendingEnable) THEN
```

```
453                                         FB_step:=90;
                                            //"Done" signal waits until motion is finished (no blending)
454                                 ELSE
455                                         CBT.WaitingReturn:=FALSE;
                                            //Flag to set the robot in "released" state to avoid other FB to be executed
456                                         CmdID_Ack:=STRING_TO_UINT(str_CommandID);  //Output the Command ID when ack is done
457                                         FB_step:=200;
                                            //"Done" signal does not wait until  motion ends (allows blending)
458                                 END_IF;
459                             ELSIF FIND(StringOfReceivedData,'ERROR') <> 0 THEN
460                                 CBT.WaitingReturn:=FALSE;
                                            //Flag to set the robot in "released" state to avoid other FB to be executed
461                                 ToError_Sts := TRUE;
462                                 ToErrorID:=16#80;
463                                 ToErrorDescrip:=' Command rejected';
464                             END_IF;
465                 END_IF;
466
467         90: (*      Clear receive buffer                *)
468                             TCP_Clear_Buffer_Exe:=TRUE;
469
470                             IF TCP_Clear_Buffer.Done THEN
471                                 TCP_Clear_Buffer_Exe:=FALSE;
472                                 FB_step:=100;
473                             END_IF;
474
475                             IF TCP_Clear_Buffer.Error THEN
476                                 ToError_Sts := TRUE;
477                                 ToErrorID:=16#90;
478                                 ToErrorDescrip:='Clear buffer error';
479                             END_IF;
480
481
482         100: (*     Request receiving data                          *)
483
484                             CmdID_Ack:=STRING_TO_UINT(str_CommandID);               //Output the Command ID when ack is done
485
486                             TCP_Rcv_TimeOut:=0;                                     //0: No timeouts
487                             TCP_Rcv_Size:=256;                                      //Set number of bytes to read from the receive buffer
488                             StringOfReceivedData:='';               //Clear the variable where Receive data array is compiled
489
490                             TCP_Rcv_Exe :=TRUE;
491
492                             IF TCP_Rcv.Done THEN
493                                 StringOfReceivedData:=AryToString(TCP_Rcv_Data[0],TCP_Rcv_Size);

                                //Converts a maximum of 1985 BYTE array to a text string (starting from index [0])
494                                 TCP_Rcv_Exe:=FALSE;
495                                 FB_step:=110;
496                             END_IF;
497
498                             IF TCP_Rcv.Error THEN
499                                 TCP_Rcv_Exe:=FALSE;
500                                 ToError_Sts := TRUE;
501                                 ToErrorID:=16#100;
502                                 ToErrorDescrip:='TCP receive error';
503                             END_IF;
504
505
506         110: (*     Check acknowledgement Motion Completed          *)
507
508                             IF FIND(StringOfReceivedData,'TMSTA') <> 0 THEN
                                //QueueTag acknowledgement
509                                 IF FIND(StringOfReceivedData,str_CommandID) <> 0 THEN               //QueueTag for the last motion command
510                                     IF FIND(StringOfReceivedData,'true') <> 0 THEN
                                        //Motion finished
```

```
511                                                     CBT.WaitingReturn:=FALSE;
                                                                        //Flag to set the robot in "released"
    state to avoid other FB to be executed
512                                                     CmdID_Ack:=STRING_TO_UINT(str_CommandID);
                    //Output the Command ID when ack is done
513                                                     FB_step:=200;
514                                                 ELSIF FIND(StringOfReceivedData,'false') <> 0 THEN
515                                                     CBT.WaitingReturn:=FALSE;
                                                                        //Flag to set the robot in "released"
    state to avoid other FB to be executed
516                                                     ToError_Sts := TRUE;
517                                                     ToErrorID:=16#110;
518                                                     ToErrorDescrip:=' Motion failed';
519                                                 END_IF;
520                                         ELSE    //no str_CommandID
521                                                 FB_step:=100;
522                                         END_IF;
523                                 ELSE            //no TMSTA
524                                         FB_step:=100;
525                                 END_IF;
526
527
528         200: (*             End Execution       *)
529                     ToDone_Sts:=TRUE;
530
531         END_CASE ;
532
533     END_IF;
534
535
536
537
538     (* Function Bolcks
                                                                                                    *)
539     (* ------------------------------------------------------------------  *)
540
541
542     TCP_Clear_Buffer(
543         Execute:=TCP_Clear_Buffer_Exe,
544         Socket:=CBT.Socket
545         //Done=>, Busy=>, Error=>, ErrorID=>
546         );
547
548     TCP_Send(
549         Execute:=TCP_Send_Exe,
550         Socket:=CBT.Socket,
551         SendDat:=TCP_Send_Data[0],
552         Size:=TCP_Send_Size
553         //Done=>, Busy=>, Error=>, ErrorID=>
554         );
555
556     TCP_Rcv(
557         Execute:=TCP_Rcv_Exe,
558         Socket:=CBT.Socket,
559         TimeOut:=TCP_Rcv_TimeOut,
560         Size:=TCP_Rcv_Size,
561         RcvDat:=TCP_Rcv_Data[0],
562         //Done=>, Busy=>, Error=>, ErrorID=>,
563         RcvSize=>TCP_Rcv_RcvSize);
564
565
```

A3. CBT_MoveRelative

```
(* ************************************************************************************ *)
(*                              Robot StartUp FB                                        *)
(* ************************************************************************************ *)
(* ------------------------------------------------------------------------------------
FB name:                        CBT_MoveRelative
(* ------------------------------------------------------------------------------------
FB version:             V01
Library:                        CBT_Commands
SS version:             V1.44
Date:                           June 2021
Author:                         Ruben Sanchez Boli
Description:            Sends motion commands to relative positions
                        - move_PTP() -> Syntax 2 in expression editor
                        - move_Line() -> Syntax 2 in expression editor
(* ------------------------------------------------------------------------------------ *)


// Detect rising flag on Execute input
R_TRIG_Execute(Clk:=Execute, Q=>flagExecute);

//Sequence initialization
IF flagExecute THEN
    IF FB_status <> 2 THEN              //FBstatus_Step different than 2 - Busy (avoids re-execution)
            iParameters:=Parameters;    //Copy internal variable
            FB_status:=1;               //initates secuences
            FB_step:=10;                //Initates algorithm sequence
    END_IF;
END_IF;

//In case of EStop is triggered during FB execution
IF Execute AND NOT(CBT.Connected) THEN
    ToError_Sts := TRUE;
    ToErrorID:=16#99;
    ToErrorDescrip:='Robot not connected';
END_IF;

(* FB State Diagram control                                                        *)
(* ------------------------------------------------------------------------------- *)
//FBstatus_Step :  0 - Idle
//FBstatus_Step :  1 - Reset
//FBstatus_Step :  2 - Busy
//FBstatus_Step :  3 - Error
//FBstatus_Step :  4 - Error deactive
//FBstatus_Step :  5 - Done active
//FBstatus_Step :  6 - Done deactive


//Sequence
CASE FB_status OF

0: (*                   Idle                        *)
            FB_status:=0;

1: (*                   Reset               *)
            ToError_Sts:=FALSE;
            ToDone_Sts:=FALSE;

            Done := FALSE;
            Busy := FALSE;
            Error := FALSE;
            ErrorID:=0;
            ErrorDescrip:=' ';
            FB_status:=2;

2: (*           Busy state *)
            Done := FALSE;
            Busy := TRUE;
            Error := FALSE;
```

```
69.                 ErrorID:=0;
70.                 ErrorDescrip:=' ';
71.
72.                 IF ToError_Sts THEN
73.                         FB_status:=3;                   //Error flag
74.                 END_IF;
75.
76.                 IF ToDone_Sts THEN
77.                         FB_status:=5;                   //Done flag
78.                 END_IF;
79.
80.  3: (*         Error state *)
81.                 Done := FALSE;
82.                 Busy := FALSE;
83.                 Error := TRUE;
84.                 ErrorID:=ToErrorID;
85.                 ErrorDescrip:=ToErrorDescrip;
86.
87.                 IF NOT Execute THEN
88.                         FB_status:= 4;                                  //Returns to idle
89.                 END_IF;
90.
91.  4: (*         Error Deactive if not execute       *)
92.                 Done := FALSE;
93.                 Busy := FALSE;
94.                 Error := FALSE;
95.                 ErrorID:=ToErrorID;
96.                 ErrorDescrip:=ToErrorDescrip;
97.
98.  5: (*         Done state *)
99.                 Done := TRUE;
100.                Busy := FALSE;
101.                Error := FALSE;
102.                ErrorID:=0;
103.                ErrorDescrip:=' ';
104.
105.                IF NOT Execute THEN
106.                        FB_status:= 6;                                  //Returns to idle
107.                END_IF;
108.
109. 6: (*                  Done deactive if not Execute                *)
110.                Done := FALSE;
111.                Busy := FALSE;
112.                Error := FALSE;
113.                ErrorID:=0;
114.                ErrorDescrip:=' ';
115.
116. END_CASE ;
117.
118.
119.
120. (* FB Algorithm                                                   *)
121. (* ------------------------------------------------------------- *)
122.
123. IF Busy THEN
124.
125.     CASE FB_step OF
126.     0: (*       Idle                    *)
127.                 FB_step:=0;
128.
129.     10: (*      Initialization                  *)
130.                 //InternalStep variables
131.                 TCP_Clear_Buffer_Exe        :=FALSE;
132.                 TCP_Send_Exe                                    :=FALSE;
133.                 TCP_Rcv_Exe                                     :=FALSE;
134.                 ToErrorID                                               :=0;
135.                 ToErrorDescrip                                  :=' ';
136.                 CmdID_Ack                                               :=Parameters.CommandID;
137.
138.                 IF CBT.Connected and NOT(CBT.WaitingReturn) THEN
139.                         FB_step:=20;
```

```
140.                        ELSIF NOT(CBT.Connected) THEN
141.                            ToError_Sts := TRUE;
142.                            ToErrorID:=16#10;
143.                            ToErrorDescrip:='Robot not connected';
144.                        ELSIF (CBT.WaitingReturn) THEN
145.                            ToError_Sts := TRUE;
146.                            ToErrorID:=16#11;
147.                            ToErrorDescrip:='Other FB is under execution';
148.                        END_IF;
149.
150.
151.    20: (*      Clear receive buffer                *)
152.                        TCP_Clear_Buffer_Exe:=TRUE;
153.
154.                        IF TCP_Clear_Buffer.Done THEN
155.                            TCP_Clear_Buffer_Exe:=FALSE;
156.                            FB_step:=25;
157.                        END_IF;
158.
159.                        IF TCP_Clear_Buffer.Error THEN
160.                            TCP_Clear_Buffer_Exe:=FALSE;
161.                            ToError_Sts := TRUE;
162.                            ToErrorID:=16#20;
163.                            ToErrorDescrip:='Clear buffer error';
164.                        END_IF;
165.

166.
167.    25: (*      Do not exceed the seetable range *)
168.
169.                        IF iParameters.MovementCommand=\\CBT_Commands_Lib\eCBT_MovCmd_Movement#Line
    THEN
170.                            IF iParameters.Speed > 4500 THEN
171.                                ToError_Sts := TRUE;
172.                                ToErrorID:=16#25;
173.                                ToErrorDescrip:='Speed range in Line: 0~4500 mm/s';
174.                            END_IF;
175.                        END_IF;
176.
177.                        IF iParameters.MovementCommand=\\CBT_Commands_Lib\eCBT_MovCmd_Movement#PTP
    THEN
178.                            IF iParameters.Speed > 100 THEN
179.                                ToError_Sts := TRUE;
180.                                ToErrorID:=16#26;
181.                                ToErrorDescrip:='Speed range in PTP: 0~100 %';
182.                            END_IF;
183.                        END_IF;
184.
185.                        IF (iParameters.AccelTime < 150) OR (iParameters.AccelTime > 9999) THEN
186.                            ToError_Sts := TRUE;
187.                            ToErrorID:=16#27;
188.                            ToErrorDescrip:='Acceleration Time range is: 150~9999 ms';
189.                        END_IF;
190.
191.                        IF (iParameters.CommandID < 2) OR (iParameters.CommandID > 9) THEN
192.                            ToError_Sts := TRUE;
193.                            ToErrorID:=16#28;
194.                            ToErrorDescrip:='Command ID range is: 2~9';
195.                        END_IF;
196.
197.                        FB_step:=30;
198.
199.
200.    30: (*  iParameters conversion to string *)
201.
202.                        str_CommandID:=UINT_TO_STRING(iParameters.CommandID);
203.
204.                        // iParameters.MovementCommand
205.                        IF iParameters.MovementCommand=\\CBT_Commands_Lib\eCBT_MovCmd_Movement#Line
    THEN
206.                            str_MovementCommand:='Move_Line';
```

```
207.                        END_IF;
208.
209.                        IF iParameters.MovementCommand=\\CBT_Commands_Lib\eCBT_MovCmd_Movement#PTP
    THEN
210.                            str_MovementCommand:='Move_PTP';
211.                        END_IF;
212.
213.                        //iParameters.DataFormat
214.                        IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#JPP_Rel THEN
215.                            str_DataFormat:='JPP';
216.                        END_IF;
217.
218.                        IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#CAP_Rel
    THEN
219.                            str_DataFormat:='CAP';
220.                        END_IF;
221.
222.                        IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#CAR_Rel
    THEN
223.                            str_DataFormat:='CAR';
224.                        END_IF;
225.
226.                        IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#CPP_Rel
    THEN
227.                            str_DataFormat:='CPP';
228.                        END_IF;
229.
230.                        IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#CPR_Rel
    THEN
231.                            str_DataFormat:='CPR';
232.                        END_IF;
233.
234.                        IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#TAP_Rel
    THEN
235.                            str_DataFormat:='TAP';
236.                        END_IF;
237.
238.                        IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#TAR_Rel
    THEN
239.                            str_DataFormat:='TAR';
240.                        END_IF;
241.
242.                        IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#TPP_Rel
    THEN
243.                            str_DataFormat:='TPP';
244.                        END_IF;
245.
246.                        IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdRel_DataFormat#TPR_Rel
    THEN
247.                            str_DataFormat:='TPR';
248.                        END_IF;
249.
250.
251.                        str_TP_X := RealToFormatString(In:=iParameters.TargetPosition.X, Exponent:=FALSE,
    Sign:=TRUE, MinLen:=1, DecPlace:=0);
252.                        str_TP_Y := RealToFormatString(In:=iParameters.TargetPosition.Y, Exponent:=FALSE,
    Sign:=TRUE, MinLen:=1, DecPlace:=0);
253.                        str_TP_Z := RealToFormatString(In:=iParameters.TargetPosition.Z, Exponent:=FALSE, Sign:=TRUE,
    MinLen:=1, DecPlace:=0);
254.                        str_TP_RX := RealToFormatString(In:=iParameters.TargetPosition.RX, Exponent:=FALSE,
    Sign:=TRUE, MinLen:=1, DecPlace:=0);
255.                        str_TP_RY := RealToFormatString(In:=iParameters.TargetPosition.RY, Exponent:=FALSE,
    Sign:=TRUE, MinLen:=1, DecPlace:=0);
256.                        str_TP_RZ := RealToFormatString(In:=iParameters.TargetPosition.RZ, Exponent:=FALSE,
    Sign:=TRUE, MinLen:=1, DecPlace:=0);
257.                        str_TP1:=CONCAT(str_TP_X,',',str_TP_Y,',',str_TP_Z);
258.                        str_TP2:=CONCAT(str_TP_RX,',',str_TP_RY,',',str_TP_RZ);
259.                        str_TargetPosition:=CONCAT(str_TP1,',',str_TP2);
260.
261.                        str_Speed:=UINT_TO_STRING(iParameters.Speed);
262.                        str_AccelTime:=UINT_TO_STRING(iParameters.AccelTime);
```

```
263.
264.                    IF iParameters.BlendingEnable THEN
265.                        str_BlendingValue:=UINT_TO_STRING(iParameters.BlendingValue);
266.                    ELSE
267.                        str_BlendingValue:='0';
268.                    END_IF;
269.
270.                    IF iParameters.PrecisePositioning THEN
271.                        str_PrecisePositioning:='true';
272.                    ELSE
273.                        str_PrecisePositioning:='false';
274.                    END_IF;
275.
276.                    IF (str_MovementCommand='Move_PTP') THEN
277.                        IF (str_DataFormat='JPP') OR
278.                            (str_DataFormat='CPP') OR
279.                            (str_DataFormat='TPP') THEN
280.                            FB_step:=70;
281.                        ELSE
282.                            ToError_Sts := TRUE;
283.                            ToErrorID:=16#30;
284.                            ToErrorDescrip:='DataFormat invalid for PTP command';
285.                        END_IF;
286.                    END_IF;
287.
288.                    IF (str_MovementCommand='Move_Line') THEN
289.                        IF (str_DataFormat='CPP') OR
290.                            (str_DataFormat='CPR') OR
291.                            (str_DataFormat='CAP') OR
292.                            (str_DataFormat='CAR') OR
293.                            (str_DataFormat='TPP') OR
294.                            (str_DataFormat='TPR') OR
295.                            (str_DataFormat='TAP') OR
296.                            (str_DataFormat='TAR') THEN
297.                            FB_step:=70;
298.                        ELSE
299.                            ToError_Sts := TRUE;
300.                            ToErrorID:=16#31;
301.                            ToErrorDescrip:='DataFormat invalid for Line command';
302.                        END_IF;
303.                    END_IF;
304.
305.            FB_step:=40;
306.
307.    40: (*  Frame buildingfor CheckSum calculation *)
308.
309.                    Header:='TMSCT';
310.
311.                    Script_Command:=CONCAT(    CONCAT(str_MovementCommand,'(',str_DataFormat,','),
312.
                        CONCAT(str_TargetPosition,',',str_Speed,','),
313.
                        CONCAT(str_AccelTime,',',str_BlendingValue,','),
314.
                        CONCAT(str_PrecisePositioning,')') );
315.
316.                    //When no blending Motion ended acknowledgement with QueueTag()
317.                    IF NOT(iParameters.BlendingEnable) THEN
318.                        Script_Command:=CONCAT(Script_Command,'$R$LQueueTag(',str_CommandID,',0)');
319.                    END_IF;
320.
321.                    //Exits Listen Node in TMflow with ScriptExit()
322.                    IF iParameters.ExitNode THEN
323.                        Script_Command:=CONCAT(Script_Command,'$R$L','ScriptExit()');
324.                    END_IF;
325.
326.                    // CheckSum calculation
327.                    str_Length:=INT_TO_STRING(LEN(CONCAT(str_CommandID,',',Script_Command)));

            //Length for DATA command
```

```
328.                    str_Checksum_Calc :=
    CONCAT(CONCAT(Header,',',str_Length,',',str_CommandID),CONCAT(',',Script_Command),','); //Checksum
    includes: Header, Length for DATA command, str_CommandID and Script_Command
329.
330.
        Checksum_Length:=ToAryByte(str_Checksum_Calc,_eBYTE_ORDER#_LOW_HIGH,Send_Checksum[0]);
331.
332.                    IF Checksum_Length>0 THEN
333.                        FB_step:=50;
334.                    ELSE
335.                        ToError_Sts := TRUE;
336.                        ToErrorID:=16#40;
337.                        ToErrorDescrip:='Checksum length not valid';
338.                    END_IF;
339.
340.
341.
342.     50: (*  Frame buildingto send command to TMflow *)
343.
344.                    FOR i:=1 TO LEN(str_Checksum_Calc) BY 1 DO
345.                        Send_Checksum[0]:=Send_Checksum[0] XOR Send_Checksum[i];
346.                    END_FOR;
347.
348.
349.                    str_Checksum:=BYTE_TO_STRING(Send_Checksum[0]);
350.                    str_SendFrame :=
    CONCAT(CONCAT('$$',Header,',',str_Length,','),CONCAT(str_CommandID,',',Script_Command),',*',str_Checksum,'$R$L');
351.
352.                    Length:=LEN(str_SendFrame);
353.                    Long:=ToAryByte(str_SendFrame, _eBYTE_ORDER#_LOW_HIGH,TCP_Send_Data[0]);

354.
355.                    IF Long>0 THEN
356.                        FB_step:=60;
357.                    ELSE
358.                        ToError_Sts := TRUE;
359.                        ToErrorID:=16#50;
360.                        ToErrorDescrip:='Final frame building is error end';
361.                    END_IF;
362.
363.
364.     60: (*      Send command                    *)
365.
366.                    TCP_Send_Size:=LEN(str_SendFrame);
367.                    ToAryByte(In:=str_SendFrame, Order:=_eBYTE_ORDER#_LOW_HIGH,
    AryOut:=TCP_Send_Data[0]);
368.
369.                    TCP_Send_Exe        :=TRUE;
370.
371.                    IF TCP_Send.Done THEN
372.                        CBT.WaitingReturn:=TRUE;            //Flag to set the robot in "busy" state to avoid
    other FB to be executed
373.                        TCP_Send_Exe:=FALSE;
374.                        FB_step:=70;
375.                    END_IF;
376.
377.                    IF TCP_Send.Error THEN
378.                        TCP_Send_Exe:=FALSE;
379.                        ToError_Sts := TRUE;
380.                        ToErrorID:=16#60;
381.                        ToErrorDescrip:='TCP send error';
382.                    END_IF;
383.
384.
385.     70: (*      Request receiving data                  *)
386.
387.                    TCP_Rcv_TimeOut:=0;                                 //0: No timeouts
388.                    TCP_Rcv_Size:=256;                                  //Set number of bytes to read from
    the receive buffer
389.                    StringOfReceivedData:='';           //Clear the variable where Receive data array is compiled
390.
```

```
391.                    TCP_Rcv_Exe :=TRUE;
392.
393.                    IF TCP_Rcv.Done THEN
394.                            StringOfReceivedData:=AryToString(TCP_Rcv_Data[0],TCP_Rcv_Size);
    //Converts a maximum of 1985 BYTE array to a text string (starting from index [0])
395.                            TCP_Rcv_Exe:=FALSE;
396.                            FB_step:=80;
397.                    END_IF;
398.
399.                    IF TCP_Rcv.Error THEN
400.                            TCP_Rcv_Exe:=FALSE;
401.                            ToError_Sts := TRUE;
402.                            ToErrorID:=16#70;
403.                            ToErrorDescrip:='TCP receive error';
404.                    END_IF;
405.
406.
407.     80: (*     Check acknowledgement Command accepted                    *)
408.
409.             IF FIND(StringOfReceivedData,'TMSTA') <> 0 THEN                    //Message no vaild
410.                     FB_step:=70;
411.             END_IF;
412.
413.             IF FIND(StringOfReceivedData,'TMSCT') <> 0 THEN
414.                     IF FIND(StringOfReceivedData,'OK') <> 0 THEN                                    //Command
    accepted
415.                             IF NOT(iParameters.BlendingEnable) THEN
416.                                     FB_step:=90;
                                                                                    //"Done" signal waits until motion
    is finished (no blending)
417.                             ELSE
418.                                     CBT.WaitingReturn:=FALSE;
                                        //Flag to set the robot in "released" state to avoid other FB to be executed
419.                                     CmdID_Ack:=STRING_TO_UINT(str_CommandID);  //Output the Command
    ID when ack is done
420.                                     FB_step:=200;
                                                                                    //"Done" signal does not wait until
    motion ends (allows blending)
421.                             END_IF;
422.                     ELSIF FIND(StringOfReceivedData,'ERROR') <> 0 THEN
423.                             CBT.WaitingReturn:=FALSE;
                                        //Flag to set the robot in "released" state to avoid other FB to be executed
424.                             ToError_Sts := TRUE;
425.                             ToErrorID:=16#80;
426.                             ToErrorDescrip:=' Command rejected';
427.                     END_IF;
428.             END_IF;
429.
430.
431.     90: (*     Clear receive buffer            *)
432.                     TCP_Clear_Buffer_Exe:=TRUE;
433.
434.                     IF TCP_Clear_Buffer.Done THEN
435.                             TCP_Clear_Buffer_Exe:=FALSE;
436.                             FB_step:=100;
437.                     END_IF;
438.
439.                     IF TCP_Clear_Buffer.Error THEN
440.                             ToError_Sts := TRUE;
441.                             ToErrorID:=16#90;
442.                             ToErrorDescrip:='Clear buffer error';
443.                     END_IF;
444.
445.
446.     100: (*    Request receiving data                  *)
447.
448.                     CmdID_Ack:=STRING_TO_UINT(str_CommandID);                                      //Output the
    Command ID when ack is done
449.
450.                     TCP_Rcv_TimeOut:=0;                                    //0: No timeouts
```

```
451.                    TCP_Rcv_Size:=256;                                  //Set number of bytes to read from
    the receive buffer
452.                    StringOfReceivedData:='';           //Clear the variable where Receive data array is compiled
453.
454.                    TCP_Rcv_Exe :=TRUE;
455.
456.                    IF TCP_Rcv.Done THEN
457.                            StringOfReceivedData:=AryToString(TCP_Rcv_Data[0],TCP_Rcv_Size);
        //Converts a maximum of 1985 BYTE array to a text string (starting from index [0])
458.                            TCP_Rcv_Exe:=FALSE;
459.                            FB_step:=110;
460.                    END_IF;
461.
462.                    IF TCP_Rcv.Error THEN
463.                            TCP_Rcv_Exe:=FALSE;
464.                            ToError_Sts := TRUE;
465.                            ToErrorID:=16#100;
466.                            ToErrorDescrip:='TCP receive error';
467.                    END_IF;
468.
469.
470.        110: (*     Check acknowledgement Motion Completed          *)
471.
472.                    IF FIND(StringOfReceivedData,'TMSTA') <> 0 THEN
        //QueueTag acknowledgement
473.                            IF FIND(StringOfReceivedData,str_CommandID) <> 0 THEN           //QueueTag
    for the last motion command
474.                                    IF FIND(StringOfReceivedData,'true') <> 0 THEN
                    //Motion finished
475.                                            CBT.WaitingReturn:=FALSE;
                                                                    //Flag to set the robot in "released"
    state to avoid other FB to be executed
476.                                            CmdID_Ack:=STRING_TO_UINT(str_CommandID);
                    //Output the Command ID when ack is done
477.                                            FB_step:=200;
478.                                    ELSIF FIND(StringOfReceivedData,'false') <> 0 THEN
479.                                            CBT.WaitingReturn:=FALSE;
                                                                    //Flag to set the robot in "released"
    state to avoid other FB to be executed
480.                                            ToError_Sts := TRUE;
481.                                            ToErrorID:=16#110;
482.                                            ToErrorDescrip:=' Motion failed';
483.                                    END_IF;
484.                            ELSE    //no str_CommandID
485.                                    FB_step:=100;
486.                            END_IF;
487.                    ELSE            //no TMSTA
488.                            FB_step:=100;
489.                    END_IF;
490.
491.
492.
493.        200: (*             End Execution       *)
494.                ToDone_Sts:=TRUE;
495.
496.        END_CASE ;
497.
498. END_IF;
499.
500.
501.
502.
503. (* Function Bolcks
                                                                    *)
504. (* ------------------------------------------------------------  *)
505.
506.
507. TCP_Clear_Buffer(
508.        Execute:=TCP_Clear_Buffer_Exe,
509.        Socket:=CBT.Socket
510.        //Done=>, Busy=>, Error=>, ErrorID=>
```

```
511.        );
512.
513.  TCP_Send(
514.        Execute:=TCP_Send_Exe,
515.        Socket:=CBT.Socket,
516.        SendDat:=TCP_Send_Data[0],
517.        Size:=TCP_Send_Size
518.        //Done=>, Busy=>, Error=>, ErrorID=>
519.        );
520.
521.  TCP_Rcv(
522.        Execute:=TCP_Rcv_Exe,
523.        Socket:=CBT.Socket,
524.        TimeOut:=TCP_Rcv_TimeOut,
525.        Size:=TCP_Rcv_Size,
526.        RcvDat:=TCP_Rcv_Data[0],
527.        //Done=>, Busy=>, Error=>, ErrorID=>,
528.        RcvSize=>TCP_Rcv_RcvSize);
529.
530.
531.
532.
```

A4. CBT_MoveCircle

```
1.  (* ************************************************************************** *)
2.  (*                                Robot StartUp FB                            *)
3.  (* ************************************************************************** *)
4.  (* ---------------------------------------------------------------------------
5.  FB name:                        CBT_MoveCircle
6.  (* ---------------------------------------------------------------------------
7.  FB version:             V01
8.  Library:                CBT_Commands
9.  SS version:             V1.44
10. Date:                   June 2021
11. Author:                 Ruben Sanchez Boli
12. Description:            Sends circle motion commands
13.                         - Circle() -> Syntax 2 in expression editor
14. (* --------------------------------------------------------------------------- *)
15.
16.
17. // Detect rising flag on Execute input
18. R_TRIG_Execute(Clk:=Execute, Q=>flagExecute);
19.
20. //Sequence initialization
21. IF flagExecute THEN
22.       IF FB_status <> 2 THEN                          //FBstatus_Step different than 2 - Busy (avoids re-
    execution)
23.             iParameters:=Parameters;          //Copy internal variable
24.             FB_status:=1;                                                   //initates secuences
25.             FB_step:=10;                                                    //Initates algorithm
    sequence
26.       END_IF;
27. END_IF;
28.
29. //In case of EStop is triggered during FB execution
30. IF Execute AND NOT(CBT.Connected) THEN
31.       ToError_Sts := TRUE;
32.       ToErrorID:=16#99;
33.       ToErrorDescrip:='Robot not connected';
34. END_IF;
35.
```

```
(* FB State Diagram control                                                    *)
(* ---------------------------------------------------------------------------- *)
//FBstatus_Step :  0 - Idle
//FBstatus_Step :  1 - Reset
//FBstatus_Step :  2 - Busy
//FBstatus_Step :  3 - Error
//FBstatus_Step :  4 - Error deactive
//FBstatus_Step :  5 - Done active
//FBstatus_Step :  6 - Done deactive


//Sequence
CASE FB_status OF

0: (*                   Idle                        *)
                FB_status:=0;


1: (*                   Reset               *)
                ToError_Sts:=FALSE;
                ToDone_Sts:=FALSE;

                Done := FALSE;
                Busy := FALSE;
                Error := FALSE;
                ErrorID:=0;
                ErrorDescrip:=' ';
                FB_status:=2;


2: (*           Busy state *)
                Done := FALSE;
                Busy := TRUE;
                Error := FALSE;
                ErrorID:=0;
                ErrorDescrip:=' ';

                IF ToError_Sts THEN
                        FB_status:=3;               //Error flag
                END_IF;

                IF ToDone_Sts THEN
                        FB_status:=5;               //Done flag
                END_IF;


3: (*           Error state *)
                Done := FALSE;
                Busy := FALSE;
                Error := TRUE;
                ErrorID:=ToErrorID;
                ErrorDescrip:=ToErrorDescrip;

                IF NOT Execute THEN
                        FB_status:= 4;                                  //Returns to idle
                END_IF;

4: (*           Error Deactive if not execute       *)
                Done := FALSE;
                Busy := FALSE;
                Error := FALSE;
                ErrorID:=ToErrorID;
                ErrorDescrip:=ToErrorDescrip;


5: (*           Done state *)
                Done := TRUE;
                Busy := FALSE;
                Error := FALSE;
                ErrorID:=0;
                ErrorDescrip:=' ';
```

```
107.
108.                    IF NOT Execute THEN
109.                            FB_status:= 6;                                  //Returns to idle
110.                    END_IF;
111.
112.  6: (*                     Done deactive if not Execute                    *)
113.                    Done := FALSE;
114.                    Busy := FALSE;
115.                    Error := FALSE;
116.                    ErrorID:=0;
117.                    ErrorDescrip:=' ';
118.
119.  END_CASE ;
120.
121.
122.
123.  (* FB Algorithm                                                          *)
124.  (* ---------------------------------------------------------------------  *)
125.
126.  IF Busy THEN
127.
128.      CASE FB_step OF
129.      0: (*       Idle                    *)
130.                  FB_step:=0;
131.
132.      10: (*      Initialization                  *)
133.                  //InternalStep variables
134.                  TCP_Clear_Buffer_Exe        :=FALSE;
135.                  TCP_Send_Exe                                    :=FALSE;
136.                  TCP_Rcv_Exe                                     :=FALSE;
137.                  ToErrorID                                               :=0;
138.                  ToErrorDescrip                                  :=' ';
139.                  CmdID_Ack                                               :=Parameters.CommandID;
140.
141.                  IF CBT.Connected and NOT(CBT.WaitingReturn) THEN
142.                          FB_step:=20;
143.                  ELSIF NOT(CBT.Connected) THEN
144.                          ToError_Sts := TRUE;
145.                          ToErrorID:=16#10;
146.                          ToErrorDescrip:='Robot not connected';
147.                  ELSIF (CBT.WaitingReturn) THEN
148.                          ToError_Sts := TRUE;
149.                          ToErrorID:=16#11;
150.                          ToErrorDescrip:='Other FB is under execution';
151.                  END_IF;
152.
153.
154.      20: (*      Clear receive buffer            *)
155.                  TCP_Clear_Buffer_Exe:=TRUE;
156.
157.                  IF TCP_Clear_Buffer.Done THEN
158.                          TCP_Clear_Buffer_Exe:=FALSE;
159.                          FB_step:=25;
160.                  END_IF;
161.
162.                  IF TCP_Clear_Buffer.Error THEN
163.                          TCP_Clear_Buffer_Exe:=FALSE;
164.                          ToError_Sts := TRUE;
165.                          ToErrorID:=16#20;
166.                          ToErrorDescrip:='Clear buffer error';
167.                  END_IF;
168.

169.
170.      25: (*      Do not exceed the seetable range *)
171.
172.                  IF iParameters.Speed > 4500 THEN
173.                          ToError_Sts := TRUE;
174.                          ToErrorID:=16#25;
175.                          ToErrorDescrip:='Speed range in Line: 0~4500 mm/s';
176.                  END_IF;
```

```
177.
178.                            IF iParameters.Speed > 100 THEN
179.                                    ToError_Sts := TRUE;
180.                                    ToErrorID:=16#26;
181.                                    ToErrorDescrip:='Speed range in PTP: 0~100 %';
182.                            END_IF;
183.
184.                            IF (iParameters.AccelTime < 150) OR (iParameters.AccelTime > 9999) THEN
185.                                    ToError_Sts := TRUE;
186.                                    ToErrorID:=16#27;
187.                                    ToErrorDescrip:='Acceleration Time range is: 150~9999 ms';
188.                            END_IF;
189.
190.                            IF (iParameters.CommandID < 2) OR (iParameters.CommandID > 9) THEN
191.                                    ToError_Sts := TRUE;
192.                                    ToErrorID:=16#28;
193.                                    ToErrorDescrip:='Command ID range is: 2~9';
194.                            END_IF;
195.
196.                            FB_step:=30;
197.
198.
199.     30: (*  iParameters conversion to string *)
200.
201.                            str_CommandID:=UINT_TO_STRING(iParameters.CommandID);
202.
203.                            str_MovementCommand:='Circle';
204.
205.                            //iParameters.DataFormat
206.                            IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdCircle_DataFormat#CAP_Circle
    THEN
207.                                    str_DataFormat:='CAP';
208.                            END_IF;
209.
210.                            IF iParameters.DataFormat=\\CBT_Commands_Lib\eCBT_MovCmdCircle_DataFormat#CPP_Circle
    THEN
211.                                    str_DataFormat:='CPP';
212.                            END_IF;
213.
214.                            str_AP_X := RealToFormatString(In:=iParameters.ArcPoint.X, Exponent:=FALSE, Sign:=TRUE,
    MinLen:=1, DecPlace:=0);
215.                            str_AP_Y := RealToFormatString(In:=iParameters.ArcPoint.Y, Exponent:=FALSE, Sign:=TRUE,
    MinLen:=1, DecPlace:=0);
216.                            str_AP_Z := RealToFormatString(In:=iParameters.ArcPoint.Z, Exponent:=FALSE, Sign:=TRUE,
    MinLen:=1, DecPlace:=0);
217.                            str_AP_RX := RealToFormatString(In:=iParameters.ArcPoint.RX, Exponent:=FALSE, Sign:=TRUE,
    MinLen:=1, DecPlace:=0);
218.                            str_AP_RY := RealToFormatString(In:=iParameters.ArcPoint.RY, Exponent:=FALSE, Sign:=TRUE,
    MinLen:=1, DecPlace:=0);
219.                            str_AP_RZ := RealToFormatString(In:=iParameters.ArcPoint.RZ, Exponent:=FALSE, Sign:=TRUE,
    MinLen:=1, DecPlace:=0);
220.                            str_AP1:=CONCAT(str_AP_X,',',str_AP_Y,',',str_AP_Z);
221.                            str_AP2:=CONCAT(str_AP_RX,',',str_AP_RY,',',str_AP_RZ);
222.                            str_ArcPoint:=CONCAT(str_AP1,',',str_AP2);
223.
224.                            str_EP_X := RealToFormatString(In:=iParameters.EndPoint.X, Exponent:=FALSE, Sign:=TRUE,
    MinLen:=1, DecPlace:=0);
225.                            str_EP_Y := RealToFormatString(In:=iParameters.EndPoint.Y, Exponent:=FALSE, Sign:=TRUE,
    MinLen:=1, DecPlace:=0);
226.                            str_EP_Z := RealToFormatString(In:=iParameters.EndPoint.Z, Exponent:=FALSE, Sign:=TRUE,
    MinLen:=1, DecPlace:=0);
227.                            str_EP_RX := RealToFormatString(In:=iParameters.EndPoint.RX, Exponent:=FALSE, Sign:=TRUE,
    MinLen:=1, DecPlace:=0);
228.                            str_EP_RY := RealToFormatString(In:=iParameters.EndPoint.RY, Exponent:=FALSE, Sign:=TRUE,
    MinLen:=1, DecPlace:=0);
229.                            str_EP_RZ := RealToFormatString(In:=iParameters.EndPoint.RZ, Exponent:=FALSE, Sign:=TRUE,
    MinLen:=1, DecPlace:=0);
230.                            str_EP1:=CONCAT(str_EP_X,',',str_EP_Y,',',str_EP_Z);
231.                            str_EP2:=CONCAT(str_EP_RX,',',str_EP_RY,',',str_EP_RZ);
232.                            str_EndPoint:=CONCAT(str_EP1,',',str_EP2);
233.
```

```
234.                        str_Speed:=UINT_TO_STRING(iParameters.Speed);
235.                        str_AccelTime:=UINT_TO_STRING(iParameters.AccelTime);
236.
237.                        IF iParameters.BlendingEnable THEN
238.                                str_BlendingValue:=UINT_TO_STRING(iParameters.BlendingValue);
239.                        ELSE
240.                                str_BlendingValue:='0';
241.                        END_IF;
242.
243.                        str_AcrAngle:=UINT_TO_STRING(iParameters.ArcAngle);
244.
245.                        IF iParameters.PrecisePositioning THEN
246.                                str_PrecisePositioning:='true';
247.                        ELSE
248.                                str_PrecisePositioning:='false';
249.                        END_IF;
250.
251.                FB_step:=40;
252.
253.        40: (*  Frame buildingfor CheckSum calculation *)
254.
255.                        Header:='TMSCT';
256.
257.                        Script_Command:=CONCAT(     CONCAT(str_MovementCommand,'(',str_DataFormat,','),
258.
                            CONCAT(str_ArcPoint,','),
259.
                            CONCAT(str_EndPoint,',',str_Speed,','),
260.
                            CONCAT(str_AccelTime,',',str_BlendingValue,','),
261.
                            CONCAT(str_AcrAngle,',',str_PrecisePositioning,')') );
262.
263.
264.                        //When no blending Motion ended acknowledgement with QueueTag()
265.                        IF NOT(iParameters.BlendingEnable) THEN
266.                                Script_Command:=CONCAT(Script_Command,'$R$LQueueTag(',str_CommandID,',0)');
267.                        END_IF;
268.
269.                        //Exits Listen Node in TMflow with ScriptExit()
270.                        IF iParameters.ExitNode THEN
271.                                Script_Command:=CONCAT(Script_Command,'$R$L','ScriptExit()');
272.                        END_IF;
273.
274.                        // CheckSum calculation
275.                        str_Length:=INT_TO_STRING(LEN(CONCAT(str_CommandID,',',Script_Command)));

                    //Length for DATA command
276.                        str_Checksum_Calc :=
    CONCAT(CONCAT(Header,',',str_Length,',',str_CommandID),CONCAT(',',Script_Command),',');                    //Checksum
    includes: Header, Length for DATA command, str_CommandID and Script_Command
277.
278.
        Checksum_Length:=ToAryByte(str_Checksum_Calc,_eBYTE_ORDER#_LOW_HIGH,Send_Checksum[0]);
279.
280.                        IF Checksum_Length>0 THEN
281.                                FB_step:=50;
282.                        ELSE
283.                                ToError_Sts := TRUE;
284.                                ToErrorID:=16#40;
285.                                ToErrorDescrip:='Checksum length not valid';
286.                        END_IF;
287.
288.
289.
290.        50: (*  Frame buildingto send command to TMflow *)
291.
292.                        FOR i:=1 TO LEN(str_Checksum_Calc) BY 1 DO
293.                                Send_Checksum[0]:=Send_Checksum[0] XOR Send_Checksum[i];
294.                        END_FOR;
295.
```

```
296.
297.                             str_Checksum:=BYTE_TO_STRING(Send_Checksum[0]);
298.                             str_SendFrame :=
    CONCAT(CONCAT('$$',Header,',',str_Length,','),CONCAT(str_CommandID,',',Script_Command),',*',str_Checksum,'$R$L');
299.
300.                             Length:=LEN(str_SendFrame);
301.                             Long:=ToAryByte(str_SendFrame, _eBYTE_ORDER#_LOW_HIGH,TCP_Send_Data[0]);

302.
303.                             IF Long>0 THEN
304.                                     FB_step:=60;
305.                             ELSE
306.                                     ToError_Sts := TRUE;
307.                                     ToErrorID:=16#50;
308.                                     ToErrorDescrip:='Final frame building is error end';
309.                             END_IF;
310.
311.
312.     60: (*      Send command                    *)
313.
314.                             TCP_Send_Size:=LEN(str_SendFrame);
315.                             ToAryByte(In:=str_SendFrame, Order:=_eBYTE_ORDER#_LOW_HIGH,
    AryOut:=TCP_Send_Data[0]);
316.
317.                             TCP_Send_Exe        :=TRUE;
318.
319.                             IF TCP_Send.Done THEN
320.                                     CBT.WaitingReturn:=TRUE;                //Flag to set the robot in "busy" state to avoid
    other FB to be executed
321.                                     TCP_Send_Exe:=FALSE;
322.                                     FB_step:=70;
323.                             END_IF;
324.
325.                             IF TCP_Send.Error THEN
326.                                     TCP_Send_Exe:=FALSE;
327.                                     ToError_Sts := TRUE;
328.                                     ToErrorID:=16#60;
329.                                     ToErrorDescrip:='TCP send error';
330.                             END_IF;
331.
332.
333.     70: (*      Request receiving data                  *)
334.
335.                             TCP_Rcv_TimeOut:=0;                                     //0: No timeouts
336.                             TCP_Rcv_Size:=256;                                      //Set number of bytes to read from
    the receive buffer
337.                             StringOfReceivedData:='';               //Clear the variable where Receive data array is compiled
338.
339.                             TCP_Rcv_Exe :=TRUE;
340.
341.                             IF TCP_Rcv.Done THEN
342.                                     StringOfReceivedData:=AryToString(TCP_Rcv_Data[0],TCP_Rcv_Size);
    //Converts a maximum of 1985 BYTE array to a text string (starting from index [0])
343.                                     TCP_Rcv_Exe:=FALSE;
344.                                     FB_step:=80;
345.                             END_IF;
346.
347.                             IF TCP_Rcv.Error THEN
348.                                     TCP_Rcv_Exe:=FALSE;
349.                                     ToError_Sts := TRUE;
350.                                     ToErrorID:=16#70;
351.                                     ToErrorDescrip:='TCP receive error';
352.                             END_IF;
353.
354.
355.     80: (*      Check acknowledgement Command accepted                  *)
356.
357.                     IF FIND(StringOfReceivedData,'TMSTA') <> 0 THEN                 //Message no vaild
358.                             FB_step:=70;
359.                     END_IF;
360.
```

```
361.                IF FIND(StringOfReceivedData,'TMSCT') <> 0 THEN
362.                    IF FIND(StringOfReceivedData,'OK') <> 0 THEN                    //Command
    accepted
363.                        IF NOT(iParameters.BlendingEnable) THEN
364.                            FB_step:=90;
                                                                    //"Done" signal waits until motion
    is finished (no blending)
365.                        ELSE
366.                            CBT.WaitingReturn:=FALSE;
                                //Flag to set the robot in "released" state to avoid other FB to be executed
367.                            CmdID_Ack:=STRING_TO_UINT(str_CommandID);  //Output the Command
    ID when ack is done
368.                            FB_step:=200;
                                                                    //"Done" signal does not wait until
    motion ends (allows blending)
369.                        END_IF;
370.                    ELSIF FIND(StringOfReceivedData,'ERROR') <> 0 THEN
371.                        CBT.WaitingReturn:=FALSE;
                                //Flag to set the robot in "released" state to avoid other FB to be executed
372.                        ToError_Sts := TRUE;
373.                        ToErrorID:=16#80;
374.                        ToErrorDescrip:=' Command rejected';
375.                    END_IF;
376.                END_IF;
377.
378.        90: (*      Clear receive buffer                *)
379.                    TCP_Clear_Buffer_Exe:=TRUE;
380.
381.                    IF TCP_Clear_Buffer.Done THEN
382.                        TCP_Clear_Buffer_Exe:=FALSE;
383.                        FB_step:=100;
384.                    END_IF;
385.
386.                    IF TCP_Clear_Buffer.Error THEN
387.                        ToError_Sts := TRUE;
388.                        ToErrorID:=16#90;
389.                        ToErrorDescrip:='Clear buffer error';
390.                    END_IF;
391.
392.
393.        100: (*     Request receiving data                  *)
394.
395.                    CmdID_Ack:=STRING_TO_UINT(str_CommandID);                       //Output the
    Command ID when ack is done
396.
397.                    TCP_Rcv_TimeOut:=0;                                             //0: No timeouts
398.                    TCP_Rcv_Size:=256;                                              //Set number of bytes to read from
    the receive buffer
399.                    StringOfReceivedData:='';           //Clear the variable where Receive data array is compiled
400.
401.                    TCP_Rcv_Exe :=TRUE;
402.
403.                    IF TCP_Rcv.Done THEN
404.                        StringOfReceivedData:=AryToString(TCP_Rcv_Data[0],TCP_Rcv_Size);
        //Converts a maximum of 1985 BYTE array to a text string (starting from index [0])
405.                        TCP_Rcv_Exe:=FALSE;
406.                        FB_step:=110;
407.                    END_IF;
408.
409.                    IF TCP_Rcv.Error THEN
410.                        TCP_Rcv_Exe:=FALSE;
411.                        ToError_Sts := TRUE;
412.                        ToErrorID:=16#100;
413.                        ToErrorDescrip:='TCP receive error';
414.                    END_IF;
415.
416.
417.        110: (*     Check acknowledgement Motion Completed      *)
418.
419.                    IF FIND(StringOfReceivedData,'TMSTA') <> 0 THEN
        //QueueTag acknowledgement
```

```
420.                                    IF FIND(StringOfReceivedData,str_CommandID) <> 0 THEN                    //QueueTag
    for the last motion command
421.                                            IF FIND(StringOfReceivedData,'true') <> 0 THEN
                    //Motion finished
422.                                                    CBT.WaitingReturn:=FALSE;
                                                                    //Flag to set the robot in "released"
    state to avoid other FB to be executed
423.                                                    CmdID_Ack:=STRING_TO_UINT(str_CommandID);
                    //Output the Command ID when ack is done
424.                                                    FB_step:=200;
425.                                            ELSIF FIND(StringOfReceivedData,'false') <> 0 THEN
426.                                                    CBT.WaitingReturn:=FALSE;
                                                                    //Flag to set the robot in "released"
    state to avoid other FB to be executed
427.                                                    ToError_Sts := TRUE;
428.                                                    ToErrorID:=16#110;
429.                                                    ToErrorDescrip:=' Motion failed';
430.                                            END_IF;
431.                                    ELSE    //no str_CommandID
432.                                            FB_step:=100;
433.                                    END_IF;
434.                            ELSE            //no TMSTA
435.                                    FB_step:=100;
436.                            END_IF;
437.
438.
439.
440.     200: (*                End Execution           *)
441.                     ToDone_Sts:=TRUE;
442.
443.     END_CASE ;
444.
445. END_IF;
446.
447.
448.
449.
450. (* Function Bolcks                                                *)
451. (* ---------------------------------------------------------------- *)
452.
453.
454. TCP_Clear_Buffer(
455.     Execute:=TCP_Clear_Buffer_Exe,
456.     Socket:=CBT.Socket
457.     //Done=>, Busy=>, Error=>, ErrorID=>
458.     );
459.
460. TCP_Send(
461.     Execute:=TCP_Send_Exe,
462.     Socket:=CBT.Socket,
463.     SendDat:=TCP_Send_Data[0],
464.     Size:=TCP_Send_Size
465.     //Done=>, Busy=>, Error=>, ErrorID=>
466.     );
467.
468. TCP_Rcv(
469.     Execute:=TCP_Rcv_Exe,
470.     Socket:=CBT.Socket,
471.     TimeOut:=TCP_Rcv_TimeOut,
472.     Size:=TCP_Rcv_Size,
473.     RcvDat:=TCP_Rcv_Data[0],
474.     //Done=>, Busy=>, Error=>, ErrorID=>,
475.     RcvSize=>TCP_Rcv_RcvSize);
476.
477.
478.
479.
```

A5. CBT_ChangeBase

```
(* ---------------------------------------------------------------------------
FB name:                CBT_ChangeBase
(* ---------------------------------------------------------------------------
FB version:             V01
Library:                CBT_Commands
SS version:             V1.44
Date:                   June 2021
Author:                 Ruben Sanchez Boli
Description:            Sets the User Coordinate System
                        - ChangeBase() -> syntax 3 in expression editor
(* --------------------------------------------------------------------------- *)


// Detect rising flag on Execute input
R_TRIG_Execute(Clk:=Execute, Q=>flagExecute);

//Sequence initialization
IF flagExecute THEN
    IF FB_status <> 2 THEN              //FBstatus_Step different than 2 - Busy (avoids re-execution)
            iParameters:=Parameters;    //Copy internal variable
            FB_status:=1;               //initates secuences
            FB_step:=10;                //Initates algorithm sequence
    END_IF;
END_IF;

//In case of EStop is triggered during FB execution
IF Execute AND NOT(CBT.Connected) THEN
    ToError_Sts := TRUE;
    ToErrorID:=16#99;
    ToErrorDescrip:='Robot not connected';
END_IF;

(* FB State Diagram control                                                *)
(* ---------------------------------------------------------------------- *)
//FBstatus_Step :  0 - Idle
//FBstatus_Step :  1 - Reset
//FBstatus_Step :  2 - Busy
//FBstatus_Step :  3 - Error
//FBstatus_Step :  4 - Error deactive
//FBstatus_Step :  5 - Done active
//FBstatus_Step :  6 - Done deactive


//Sequence
CASE FB_status OF

0: (* Idle                          *)
        FB_status:=0;


1: (*Reset              *)
        ToError_Sts:=FALSE;
        ToDone_Sts:=FALSE;

        Done := FALSE;
        Busy := FALSE;
        Error := FALSE;
        ErrorID:=0;
        ErrorDescrip:=' ';
        FB_status:=2;


2: (*Busy state *)
        Done := FALSE;
        Busy := TRUE;
        Error := FALSE;
        ErrorID:=0;
        ErrorDescrip:=' ';
```

```
69
70                  IF ToError_Sts THEN
71                          FB_status:=3;                   //Error flag
72                  END_IF;
73
74                  IF ToDone_Sts THEN
75                          FB_status:=5;                   //Done flag
76                  END_IF;
77
78
79  3: (*Error state *)
80                  Done := FALSE;
81                  Busy := FALSE;
82                  Error := TRUE;
83                  ErrorID:=ToErrorID;
84                  ErrorDescrip:=ToErrorDescrip;
85
86                  IF NOT Execute THEN
87                          FB_status:= 4;                                  //Returns to idle
88                  END_IF;
89
90  4: (*Error Deactive if not execute      *)
91                  Done := FALSE;
92                  Busy := FALSE;
93                  Error := FALSE;
94                  ErrorID:=ToErrorID;
95                  ErrorDescrip:=ToErrorDescrip;
96
97
98  5: (*Done state *)
99                  Done := TRUE;
100                 Busy := FALSE;
101                 Error := FALSE;
102                 ErrorID:=0;
103                 ErrorDescrip:=' ';
104
105                 IF NOT Execute THEN
106                         FB_status:= 6;                                  //Returns to idle
107                 END_IF;
108
109 6: (*Done deactive if not Execute              *)
110                 Done := FALSE;
111                 Busy := FALSE;
112                 Error := FALSE;
113                 ErrorID:=0;
114                 ErrorDescrip:=' ';
115
116 END_CASE ;
117
118
119
120 (* FB Algorithm                                           *)
121 (* ---------------------------------------------------------- *)
122
123 IF Busy THEN
124
125     CASE FB_step OF
126     0: (*       Idle                    *)
127                 FB_step:=0;
128
129     10: (*      Initialization                  *)
130                 //InternalStep variables
131                 TCP_Clear_Buffer_Exe        :=FALSE;
132                 TCP_Send_Exe                                    :=FALSE;
133                 TCP_Rcv_Exe                                     :=FALSE;
134                 ToErrorID                                               :=0;
135                 ToErrorDescrip                                  :=' ';
136
137                 IF CBT.Connected and NOT(CBT.WaitingReturn) THEN
138                         FB_step:=20;
139                 ELSIF NOT(CBT.Connected) THEN
```

```
140                                     ToError_Sts := TRUE;
141                                     ToErrorID:=16#10;
142                                     ToErrorDescrip:='Robot not connected';
143                             ELSIF (CBT.WaitingReturn) THEN
144                                     ToError_Sts := TRUE;
145                                     ToErrorID:=16#11;
146                                     ToErrorDescrip:='Other FB is under execution';
147                             END_IF;
148
149         20: (*      Clear receive buffer                *)
150                             TCP_Clear_Buffer_Exe:=TRUE;
151
152                             IF TCP_Clear_Buffer.Done THEN
153                                     TCP_Clear_Buffer_Exe:=FALSE;
154                                     FB_step:=30;
155                             END_IF;
156
157                             IF TCP_Clear_Buffer.Error THEN
158                                     TCP_Clear_Buffer_Exe:=FALSE;
159                                     ToError_Sts := TRUE;
160                                     ToErrorID:=16#20;
161                                     ToErrorDescrip:='Clear buffer error';
162                             END_IF;
163
164
165
166         30: (*  iParam conversion to string *)
167
168                             str_CommandID:=UINT_TO_STRING(iParameters.CommandID);
169
170                             str_TP_X := RealToFormatString(In:=iParameters.Transformation.X, Exponent:=FALSE,
    Sign:=TRUE, MinLen:=1, DecPlace:=0);
171                             str_TP_Y := RealToFormatString(In:=iParameters.Transformation.Y, Exponent:=FALSE,
    Sign:=TRUE, MinLen:=1, DecPlace:=0);
172                             str_TP_Z := RealToFormatString(In:=iParameters.Transformation.Z, Exponent:=FALSE,
    Sign:=TRUE, MinLen:=1, DecPlace:=0);
173                             str_TP_RX := RealToFormatString(In:=iParameters.Transformation.RX, Exponent:=FALSE,
    Sign:=TRUE, MinLen:=1, DecPlace:=0);
174                             str_TP_RY := RealToFormatString(In:=iParameters.Transformation.RY, Exponent:=FALSE,
    Sign:=TRUE, MinLen:=1, DecPlace:=0);
175                             str_TP_RZ := RealToFormatString(In:=iParameters.Transformation.RZ, Exponent:=FALSE,
    Sign:=TRUE, MinLen:=1, DecPlace:=0);
176                             str_TP1:=CONCAT(str_TP_X,',',str_TP_Y,',',str_TP_Z);
177                             str_TP2:=CONCAT(str_TP_RX,',',str_TP_RY,',',str_TP_RZ);
178                             str_TargetPosition:=CONCAT(str_TP1,',',str_TP2);
179
180                             FB_step:=40;
181
182         40: (*  Frame buildingfor CheckSum calculation *)
183
184                             Header:='TMSCT';
185                             str_FunctionCommand:='ChangeBase';
186                             Script_Command:=CONCAT(str_FunctionCommand,'(',str_TargetPosition,')');
187
188                             //Exits Listen Node in TMflow with ScriptExit()
189                             IF iParameters.ExitNode THEN
190                                     Script_Command:=CONCAT(Script_Command,'$R$L','ScriptExit()');
191                             END_IF;
192
193                             // CheckSum calculation
194                             str_Length:=INT_TO_STRING(LEN(CONCAT(str_CommandID,',',Script_Command)));

                    //Length for DATA command
195                             str_Checksum_Calc :=
    CONCAT(CONCAT(Header,',',str_Length,',',str_CommandID),CONCAT(',',Script_Command),',');                 //Checksum
    includes: Header, Length for DATA command, str_CommandID and Script_Command
196
197
          Checksum_Length:=ToAryByte(str_Checksum_Calc,_eBYTE_ORDER#_LOW_HIGH,Send_Checksum[0]);
198
```

```
199                             IF Checksum_Length>0 THEN
200                                     FB_step:=50;
201                             ELSE
202                                     ToError_Sts := TRUE;
203                                     ToErrorID:=16#40;
204                                     ToErrorDescrip:='Checksum length not valid';
205                             END_IF;
206
207
208
209     50: (*  Frame buildingto send command to TMflow *)
210
211                             FOR i:=1 TO LEN(str_Checksum_Calc) BY 1 DO
212                                     Send_Checksum[0]:=Send_Checksum[0] XOR Send_Checksum[i];
213                             END_FOR;
214
215                             str_Checksum:=BYTE_TO_STRING(Send_Checksum[0]);
216                             str_SendFrame :=
CONCAT(CONCAT('$$',Header,',',str_Length,','),CONCAT(str_CommandID,',',Script_Command),',*',str_Checksum,'$R$L');
217
218                             Length:=LEN(str_SendFrame);
219                             Long:=ToAryByte(str_SendFrame, _eBYTE_ORDER#_LOW_HIGH,TCP_Send_Data[0]);

220
221                             IF Long>0 THEN
222                                     FB_step:=60;
223                             ELSE
224                                     ToError_Sts := TRUE;
225                                     ToErrorID:=16#50;
226                                     ToErrorDescrip:='Final frame building is error end';
227                             END_IF;
228
229
230     60: (*      Send command                    *)
231
232                             TCP_Send_Size:=LEN(str_SendFrame);
233                             ToAryByte(In:=str_SendFrame, Order:=_eBYTE_ORDER#_LOW_HIGH,
AryOut:=TCP_Send_Data[0]);
234
235                             TCP_Send_Exe        :=TRUE;
236
237                             IF TCP_Send.Done THEN
238                                     CBT.WaitingReturn:=TRUE;                //Flag to set the robot in "busy" state to avoid
other FB to be executed
239                                     TCP_Send_Exe:=FALSE;
240                                     FB_step:=70;
241                             END_IF;
242
243                             IF TCP_Send.Error THEN
244                                     TCP_Send_Exe:=FALSE;
245                                     ToError_Sts := TRUE;
246                                     ToErrorID:=16#60;
247                                     ToErrorDescrip:='TCP send error';
248                             END_IF;
249
250
251     70: (*      Request receiving data                  *)
252
253                             TCP_Rcv_TimeOut:=0;                                     //0: No timeouts
254                             TCP_Rcv_Size:=256;                                      //Set number of bytes to read from
the receive buffer
255                             StringOfReceivedData:='';               //Clear the variable where Receive data array is compiled
256
257                             TCP_Rcv_Exe :=TRUE;
258
259                             IF TCP_Rcv.Done THEN
260                                     StringOfReceivedData:=AryToString(TCP_Rcv_Data[0],TCP_Rcv_Size);
    //Converts a maximum of 1985 BYTE array to a text string (starting from index [0])
261                                     TCP_Rcv_Exe:=FALSE;
262                                     FB_step:=80;
263                             END_IF;
```

```
264
265                             IF TCP_Rcv.Error THEN
266                                     TCP_Rcv_Exe:=FALSE;
267                                     ToError_Sts := TRUE;
268                                     ToErrorID:=16#70;
269                                     ToErrorDescrip:='TCP receive error';
270                             END_IF;
271
272
273         80: (*      Check acknowledgement Command accepted                      *)
274
275                     IF FIND(StringOfReceivedData,'OK') <> 0 THEN                        //Command accepted
276                             CBT.WaitingReturn:=FALSE;
                                //Flag to set the robot in "released" state to avoid other FB to be executed
277                             FB_step:=90;
                                                                        //Done FB when motion is finished (no blending)
278                     ELSIF FIND(StringOfReceivedData,'ERROR') <> 0 THEN
279                             CBT.WaitingReturn:=FALSE;
                                //Flag to set the robot in "released" state to avoid other FB to be executed
280                             ToError_Sts := TRUE;
281                             ToErrorID:=16#80;
282                             ToErrorDescrip:=' Command rejected';
283                     END_IF;
284
285
286         90: (*              End Execution       *)
287                     ToDone_Sts:=TRUE;
288
289         END_CASE ;
290
291     END_IF;
292
293
294
295
296     (* Function Blocks                                                  *)
297     (* ---------------------------------------------------------------- *)
298
299
300     TCP_Clear_Buffer(
301         Execute:=TCP_Clear_Buffer_Exe,
302         Socket:=CBT.Socket
303         //Done=>, Busy=>, Error=>, ErrorID=>
304         );
305
306     TCP_Send(
307         Execute:=TCP_Send_Exe,
308         Socket:=CBT.Socket,
309         SendDat:=TCP_Send_Data[0],
310         Size:=TCP_Send_Size
311         //Done=>, Busy=>, Error=>, ErrorID=>
312         );
313
314     TCP_Rcv(
315         Execute:=TCP_Rcv_Exe,
316         Socket:=CBT.Socket,
317         TimeOut:=TCP_Rcv_TimeOut,
318         Size:=TCP_Rcv_Size,
319         RcvDat:=TCP_Rcv_Data[0],
320         //Done=>, Busy=>, Error=>, ErrorID=>,
321         RcvSize=>TCP_Rcv_RcvSize);
```

A6. CBT_ChangeTCP

```
(* ---------------------------------------------------------------------------
FB name:                CBT_ChangeTCP
(* ---------------------------------------------------------------------------
FB version:             V01
Library:                CBT_Commands
SS version:             V1.44
Date:                   June 2021
Author:                 Ruben Sanchez Boli
Description:            Sets the Tool Coordinate System
                        - ChangeTCP() -> Syntax 7 in expression editor
(* --------------------------------------------------------------------------- *)


// Detect rising flag on Execute input
R_TRIG_Execute(Clk:=Execute, Q=>flagExecute);

//Sequence initialization
IF flagExecute THEN
    IF FB_status <> 2 THEN                  //FBstatus_Step different than 2 - Busy (avoids re-execution)
            iParameters:=Parameters;        //Copy internal variable
            FB_status:=1;                   //initates secuences
            FB_step:=10;                    //Initates algorithm sequence
    END_IF;
END_IF;

//In case of EStop is triggered during FB execution
IF Execute AND NOT(CBT.Connected) THEN
    ToError_Sts := TRUE;
    ToErrorID:=16#99;
    ToErrorDescrip:='Robot not connected';
END_IF;

(* FB State Diagram control                                                 *)
(* ------------------------------------------------------------------------ *)
//FBstatus_Step :  0 - Idle
//FBstatus_Step :  1 - Reset
//FBstatus_Step :  2 - Busy
//FBstatus_Step :  3 - Error
//FBstatus_Step :  4 - Error deactive
//FBstatus_Step :  5 - Done active
//FBstatus_Step :  6 - Done deactive


//Sequence
CASE FB_status OF

0: (*           Idle                    *)
            FB_status:=0;


1: (*           Rese                    *)
            ToError_Sts:=FALSE;
            ToDone_Sts:=FALSE;

            Done := FALSE;
            Busy := FALSE;
            Error := FALSE;
            ErrorID:=0;
            ErrorDescrip:=' ';
            FB_status:=2;


2: (*           Busy state *)
            Done := FALSE;
            Busy := TRUE;
            Error := FALSE;
            ErrorID:=0;
            ErrorDescrip:=' ';
```

```
69
70                  IF ToError_Sts THEN
71                          FB_status:=3;                   //Error flag
72                  END_IF;
73
74                  IF ToDone_Sts THEN
75                          FB_status:=5;                   //Done flag
76                  END_IF;
77
78
79      3: (*       Error state *)
80                  Done := FALSE;
81                  Busy := FALSE;
82                  Error := TRUE;
83                  ErrorID:=ToErrorID;
84                  ErrorDescrip:=ToErrorDescrip;
85
86                  IF NOT Execute THEN
87                          FB_status:= 4;                                  //Returns to idle
88                  END_IF;
89
90      4: (*       Error Deactive if not execute       *)
91                  Done := FALSE;
92                  Busy := FALSE;
93                  Error := FALSE;
94                  ErrorID:=ToErrorID;
95                  ErrorDescrip:=ToErrorDescrip;
96
97
98      5: (*       Done state *)
99                  Done := TRUE;
100                 Busy := FALSE;
101                 Error := FALSE;
102                 ErrorID:=0;
103                 ErrorDescrip:=' ';
104
105                 IF NOT Execute THEN
106                         FB_status:= 6;                                  //Returns to idle
107                 END_IF;
108
109     6: (*               Done deactive if not Execute            *)
110                 Done := FALSE;
111                 Busy := FALSE;
112                 Error := FALSE;
113                 ErrorID:=0;
114                 ErrorDescrip:=' ';
115
116     END_CASE ;
117
118
119
120     (* FB Algorithm                                                *)
121     (* ----------------------------------------------------------- *)
122
123     IF Busy THEN
124
125         CASE FB_step OF
126         0: (*       Idle                    *)
127                         FB_step:=0;
128
129         10: (*      Initialization                  *)
130                         //InternalStep variables
131                         TCP_Clear_Buffer_Exe        :=FALSE;
132                         TCP_Send_Exe                :=FALSE;
133                         TCP_Rcv_Exe                 :=FALSE;
134                         ToErrorID                   :=0;
135                         ToErrorDescrip              :=' ';
136
137                         IF CBT.Connected and NOT(CBT.WaitingReturn) THEN
138                                 FB_step:=20;
139                         ELSIF NOT(CBT.Connected) THEN
```

```
140                                     ToError_Sts := TRUE;
141                                     ToErrorID:=16#10;
142                                     ToErrorDescrip:='Robot not connected';
143                             ELSIF (CBT.WaitingReturn) THEN
144                                     ToError_Sts := TRUE;
145                                     ToErrorID:=16#11;
146                                     ToErrorDescrip:='Other FB is under execution';
147                             END_IF;
148
149
150     20: (*      Clear receive buffer                *)
151                             TCP_Clear_Buffer_Exe:=TRUE;
152
153                             IF TCP_Clear_Buffer.Done THEN
154                                     TCP_Clear_Buffer_Exe:=FALSE;
155                                     FB_step:=25;
156                             END_IF;
157
158                             IF TCP_Clear_Buffer.Error THEN
159                                     TCP_Clear_Buffer_Exe:=FALSE;
160                                     ToError_Sts := TRUE;
161                                     ToErrorID:=16#20;
162                                     ToErrorDescrip:='Clear buffer error';
163                             END_IF;
164
165     25: (*      Do not exceed the seetable range *)
166
167                             IF iParameters.Weight > 14 THEN
168                                     ToError_Sts := TRUE;
169                                     ToErrorID:=16#25;
170                                     ToErrorDescrip:='Weight range: 0~14 kg';
171                             END_IF;
172
173                             FB_step:=30;
174
175     30: (*  iBaseParam conversion to string *)
176
177                             str_CommandID:=UINT_TO_STRING(iParameters.CommandID);
178
179                             //TCPOffset
180                             str_TO_X := RealToFormatString(In:=iParameters.TCPOffset.X, Exponent:=FALSE, Sign:=TRUE,
    MinLen:=1, DecPlace:=0);
181                             str_TO_Y := RealToFormatString(In:=iParameters.TCPOffset.Y, Exponent:=FALSE, Sign:=TRUE,
    MinLen:=1, DecPlace:=0);
182                             str_TO_Z := RealToFormatString(In:=iParameters.TCPOffset.Z, Exponent:=FALSE, Sign:=TRUE,
    MinLen:=1, DecPlace:=0);
183                             str_TO_RX := RealToFormatString(In:=iParameters.TCPOffset.RX, Exponent:=FALSE,
    Sign:=TRUE, MinLen:=1, DecPlace:=0);
184                             str_TO_RY := RealToFormatString(In:=iParameters.TCPOffset.RY, Exponent:=FALSE,
    Sign:=TRUE, MinLen:=1, DecPlace:=0);
185                             str_TO_RZ := RealToFormatString(In:=iParameters.TCPOffset.RZ, Exponent:=FALSE, Sign:=TRUE,
    MinLen:=1, DecPlace:=0);
186                             str_TO1:=CONCAT(str_TO_X,',',str_TO_Y,',',str_TO_Z);
187                             str_TO2:=CONCAT(str_TO_RX,',',str_TO_RY,',',str_TO_RZ);
188                             str_TCPOffset:=CONCAT(str_TO1,',',str_TO2);
189
190                             //MomentOfInertia
191                             str_Ixx := RealToFormatString(In:=iParameters.MomentOfInertia.Ixx, Exponent:=FALSE,
    Sign:=TRUE, MinLen:=1, DecPlace:=0);
192                             str_Iyy := RealToFormatString(In:=iParameters.MomentOfInertia.Iyy, Exponent:=FALSE,
    Sign:=TRUE, MinLen:=1, DecPlace:=0);
193                             str_Izz := RealToFormatString(In:=iParameters.MomentOfInertia.Izz, Exponent:=FALSE,
    Sign:=TRUE, MinLen:=1, DecPlace:=0);
194                             str_Inertia:=CONCAT(str_Ixx,',',str_Iyy,',',str_Izz);
195
196                             //Weight
197                             str_Weight := RealToFormatString(In:=iParameters.Weight, Exponent:=FALSE, Sign:=TRUE,
    MinLen:=1, DecPlace:=0);
198
199                             //MassCenter
```

```
200                             str_MC_X := RealToFormatString(In:=iParameters.MassCenter.X, Exponent:=FALSE, Sign:=TRUE,
    MinLen:=1, DecPlace:=0);
201                             str_MC_Y := RealToFormatString(In:=iParameters.MassCenter.Y, Exponent:=FALSE, Sign:=TRUE,
    MinLen:=1, DecPlace:=0);
202                             str_MC_Z := RealToFormatString(In:=iParameters.MassCenter.Z, Exponent:=FALSE, Sign:=TRUE,
    MinLen:=1, DecPlace:=0);
203                             str_MC_RX := RealToFormatString(In:=iParameters.MassCenter.RX, Exponent:=FALSE,
    Sign:=TRUE, MinLen:=1, DecPlace:=0);
204                             str_MC_RY := RealToFormatString(In:=iParameters.MassCenter.RY, Exponent:=FALSE,
    Sign:=TRUE, MinLen:=1, DecPlace:=0);
205                             str_MC_RZ := RealToFormatString(In:=iParameters.MassCenter.RZ, Exponent:=FALSE,
    Sign:=TRUE, MinLen:=1, DecPlace:=0);
206                             str_MC1:=CONCAT(str_MC_X,',',str_MC_Y,',',str_MC_Z);
207                             str_MC2:=CONCAT(str_MC_RX,',',str_MC_RY,',',str_MC_RZ);
208                             str_MassCenter:=CONCAT(str_MC1,',',str_MC2);
209
210                             str_TCP1:=CONCAT(str_TCPOffset,',',str_Weight,',');
211                             str_TCP2:=CONCAT(str_Inertia,',',str_MassCenter);
212                             str_TCPCommand:=CONCAT(str_TCP1,str_TCP2);
213
214                             FB_step:=40;
215
216     40: (*  Frame buildingfor CheckSum calculation *)
217
218                             Header:='TMSCT';
219                             str_FunctionCommand:='ChangeTCP';
220                             Script_Command:=CONCAT(str_FunctionCommand,'(',str_TCPCommand,')');
221
222                             //Exits Listen Node in TMflow with ScriptExit()
223                             IF iParameters.ExitNode THEN
224                                     Script_Command:=CONCAT(Script_Command,'$R$L','ScriptExit()');
225                             END_IF;
226
227                             // CheckSum calculation
228                             str_Length:=INT_TO_STRING(LEN(CONCAT(str_CommandID,',',Script_Command)));

                        //Length for DATA command
229                             str_Checksum_Calc :=
    CONCAT(CONCAT(Header,',',str_Length,',',str_CommandID),CONCAT(',',Script_Command),','); //Checksum
    includes: Header, Length for DATA command, str_CommandID and Script_Command
230
231
        Checksum_Length:=ToAryByte(str_Checksum_Calc,_eBYTE_ORDER#_LOW_HIGH,Send_Checksum[0]);
232
233                             IF Checksum_Length>0 THEN
234                                     FB_step:=50;
235                             ELSE
236                                     ToError_Sts := TRUE;
237                                     ToErrorID:=16#40;
238                                     ToErrorDescrip:='Checksum length not valid';
239                             END_IF;
240
241
242
243     50: (*  Frame buildingto send command to TMflow *)
244
245                             FOR i:=1 TO LEN(str_Checksum_Calc) BY 1 DO
246                                     Send_Checksum[0]:=Send_Checksum[0] XOR Send_Checksum[i];
247                             END_FOR;
248
249
250                             str_Checksum:=BYTE_TO_STRING(Send_Checksum[0]);
251                             str_SendFrame :=
    CONCAT(CONCAT('$$',Header,',',str_Length,',',CONCAT(str_CommandID,',',Script_Command),',*',str_Checksum,'$R$L');
252
253                             Length:=LEN(str_SendFrame);
254                             Long:=ToAryByte(str_SendFrame, _eBYTE_ORDER#_LOW_HIGH,TCP_Send_Data[0]);

255
256                             IF Long>0 THEN
257                                     FB_step:=60;
```

```
258                             ELSE
259                                     ToError_Sts := TRUE;
260                                     ToErrorID:=16#50;
261                                     ToErrorDescrip:='Final frame building is error end';
262                             END_IF;
263
264
265     60: (*      Send command                    *)
266
267                             TCP_Send_Size:=LEN(str_SendFrame);
268                             ToAryByte(In:=str_SendFrame, Order:=_eBYTE_ORDER#_LOW_HIGH,
    AryOut:=TCP_Send_Data[0]);
269
270                             TCP_Send_Exe        :=TRUE;
271
272                             IF TCP_Send.Done THEN
273                                     CBT.WaitingReturn:=TRUE;                //Flag to set the robot in "busy" state to avoid
    other FB to be executed
274                                     TCP_Send_Exe:=FALSE;
275                                     FB_step:=70;
276                             END_IF;
277
278                             IF TCP_Send.Error THEN
279                                     TCP_Send_Exe:=FALSE;
280                                     ToError_Sts := TRUE;
281                                     ToErrorID:=16#60;
282                                     ToErrorDescrip:='TCP send error';
283                             END_IF;
284
285
286     70: (*      Request receiving data                  *)
287
288                             TCP_Rcv_TimeOut:=0;                                     //0: No timeouts
289                             TCP_Rcv_Size:=256;                                      //Set number of bytes to read from
    the receive buffer
290                             StringOfReceivedData:='';               //Clear the variable where Receive data array is compiled
291
292                             TCP_Rcv_Exe :=TRUE;
293
294                             IF TCP_Rcv.Done THEN
295                                     StringOfReceivedData:=AryToString(TCP_Rcv_Data[0],TCP_Rcv_Size);
    //Converts a maximum of 1985 BYTE array to a text string (starting from index [0])
296                                     TCP_Rcv_Exe:=FALSE;
297                                     FB_step:=80;
298                             END_IF;
299
300                             IF TCP_Rcv.Error THEN
301                                     TCP_Rcv_Exe:=FALSE;
302                                     ToError_Sts := TRUE;
303                                     ToErrorID:=16#70;
304                                     ToErrorDescrip:='TCP receive error';
305                             END_IF;
306
307
308     80: (*      Check acknowledgement Command accepted                  *)
309
310                     IF FIND(StringOfReceivedData,'OK') <> 0 THEN                    //Command accepted
311                             CBT.WaitingReturn:=FALSE;
                                //Flag to set the robot in "released" state to avoid other FB to be executed
312                             FB_step:=90;
                                                                //Done FB when motion is finished (no blending)
313                     ELSIF FIND(StringOfReceivedData,'ERROR') <> 0 THEN
314                             CBT.WaitingReturn:=FALSE;
                                //Flag to set the robot in "released" state to avoid other FB to be executed
315                             ToError_Sts := TRUE;
316                             ToErrorID:=16#80;
317                             ToErrorDescrip:=' Command rejected';
318                     END_IF;
319
320
321     90: (*              End Execution       *)
```

```
322                     ToDone_Sts:=TRUE;
323
324         END_CASE ;
325
326     END_IF;
327
328
329
330
331     (* Function Blocks                                                    *)
332     (* ------------------------------------------------------------------ *)
333
334
335     TCP_Clear_Buffer(
336         Execute:=TCP_Clear_Buffer_Exe,
337         Socket:=CBT.Socket
338         //Done=>, Busy=>, Error=>, ErrorID=>
339         );
340
341     TCP_Send(
342         Execute:=TCP_Send_Exe,
343         Socket:=CBT.Socket,
344         SendDat:=TCP_Send_Data[0],
345         Size:=TCP_Send_Size
346         //Done=>, Busy=>, Error=>, ErrorID=>
347         );
348
349     TCP_Rcv(
350         Execute:=TCP_Rcv_Exe,
351         Socket:=CBT.Socket,
352         TimeOut:=TCP_Rcv_TimeOut,
353         Size:=TCP_Rcv_Size,
354         RcvDat:=TCP_Rcv_Data[0],
355         //Done=>, Busy=>, Error=>, ErrorID=>,
356         RcvSize=>TCP_Rcv_RcvSize);
```

A7. CBT_ProgramControl

```
1     (* ---------------------------------------------------------------------------
2     FB name:                CBT_ProgramControl
3     (* ---------------------------------------------------------------------------
4     FB version:             V01
5     Library:                CBT_Commands
6     SS version:             V1.44
7     Date:                   June 2021
8     Author:                 Ruben Sanchez Boli
9     Description:            Sets the User Coordinate System
10    (* --------------------------------------------------------------------------- *)
11
12
13    // Detect rising flag on Execute input
14    R_TRIG_Execute(Clk:=Execute, Q=>flagExecute);
15
16    //Sequence initialization
17    IF flagExecute THEN
18        IF FB_status <> 2 THEN                  //FBstatus_Step different than 2 - Busy (avoids re-execution)
19                iParameters:=Parameters;        //Copy internal variable
20                FB_status:=1;                   //initates secuences
21                FB_step:=10;
22        END_IF;
23    END_IF;
24
25    //In case of EStop is triggered during FB execution
26    IF Execute AND NOT(CBT.Connected) THEN
27        ToError_Sts := TRUE;
```

```
28          ToErrorID:=16#99;
29          ToErrorDescrip:='Robot not connected';
30      END_IF;
31
32      (* FB State Diagram control                                                    *)
33      (* ---------------------------------------------------------------------------- *)
34      //FBstatus_Step :  0 - Idle
35      //FBstatus_Step :  1 - Reset
36      //FBstatus_Step :  2 - Busy
37      //FBstatus_Step :  3 - Error
38      //FBstatus_Step :  4 - Error deactive
39      //FBstatus_Step :  5 - Done active
40      //FBstatus_Step :  6 - Done deactive
41
42
43      //Sequence
44      CASE FB_status OF
45
46      0: (*                       Idle                                *)
47                  FB_status:=0;
48
49
50      1: (*                       Reset               *)
51                  ToError_Sts:=FALSE;
52                  ToDone_Sts:=FALSE;
53
54                  Done := FALSE;
55                  Busy := FALSE;
56                  Error := FALSE;
57                  ErrorID:=0;
58                  ErrorDescrip:=' ';
59                  FB_status:=2;
60
61
62      2: (*       Busy state *)
63                  Done := FALSE;
64                  Busy := TRUE;
65                  Error := FALSE;
66                  ErrorID:=0;
67                  ErrorDescrip:=' ';
68
69                  IF ToError_Sts THEN
70                              FB_status:=3;               //Error flag
71                  END_IF;
72
73                  IF ToDone_Sts THEN
74                              FB_status:=5;               //Done flag
75                  END_IF;
76
77
78      3: (*       Error state *)
79                  Done := FALSE;
80                  Busy := FALSE;
81                  Error := TRUE;
82                  ErrorID:=ToErrorID;
83                  ErrorDescrip:=ToErrorDescrip;
84
85                  IF NOT Execute THEN
86                              FB_status:= 4;                                  //Returns to idle
87                  END_IF;
88
89      4: (*       Error Deactive if not execute       *)
90                  Done := FALSE;
91                  Busy := FALSE;
92                  Error := FALSE;
93                  ErrorID:=ToErrorID;
94                  ErrorDescrip:=ToErrorDescrip;
95
96
97      5: (*       Done state *)
98                  Done := TRUE;
```

```
99                  Busy := FALSE;
100                 Error := FALSE;
101                 ErrorID:=0;
102                 ErrorDescrip:=' ';
103
104                 IF NOT Execute THEN
105                         FB_status:= 6;                                  //Returns to idle
106                 END_IF;
107
108 6: (*                  Done deactive if not Execute            *)
109                 Done := FALSE;
110                 Busy := FALSE;
111                 Error := FALSE;
112                 ErrorID:=0;
113                 ErrorDescrip:=' ';
114
115 END_CASE ;
116
117
118
119 (* FB Algorithm                                                 *)
120 (* ---------------------------------------------------------- *)
121
122 IF Busy THEN
123
124     CASE FB_step OF
125     0: (*       Idle                    *)
126                 FB_step:=0;
127
128     10: (*      Initialization                  *)
129                 //InternalStep variables
130                 TCP_Clear_Buffer_Exe        :=FALSE;
131                 TCP_Send_Exe                        :=FALSE;
132                 TCP_Rcv_Exe                         :=FALSE;
133                 ToErrorID                           :=0;
134                 ToErrorDescrip                      :=' ';
135
136                 IF CBT.Connected THEN
137                         FB_step:=20;
138                 ELSE
139                         ToError_Sts := TRUE;
140                         ToErrorID:=16#10;
141                         ToErrorDescrip:='Robot not connected';
142                 END_IF;
143
144
145     20: (*      Clear receive buffer            *)
146                 TCP_Clear_Buffer_Exe:=TRUE;
147
148                 IF TCP_Clear_Buffer.Done THEN
149                         TCP_Clear_Buffer_Exe:=FALSE;
150                         FB_step:=30;
151                 END_IF;
152
153                 IF TCP_Clear_Buffer.Error THEN
154                         TCP_Clear_Buffer_Exe:=FALSE;
155                         ToError_Sts := TRUE;
156                         ToErrorID:=16#20;
157                         ToErrorDescrip:='Clear buffer error';
158                 END_IF;
159

160
161
162     30: (*  iParam conversion to string *)
163
164                 str_CommandID:=UINT_TO_STRING(iParameters.CommandID);
165
166                 // iParameters.MovementCommand
167                 IF iParameters.Command=\\CBT_Commands_Lib\eCBT_PrgControlCmd#Pause THEN
168                         str_FunctionCommand:='Pause()';
```

```
169                             END_IF;
170
171                             IF iParameters.Command=\\CBT_Commands_Lib\eCBT_PrgControlCmd#Resume THEN
172                                     str_FunctionCommand:='Resume()';
173                             END_IF;
174
175                             FB_step:=40;
176
177         40: (*  Frame buildingfor CheckSum calculation *)
178
179                             Header:='TMSCT';
180
181                             Script_Command:=str_FunctionCommand;
182
183                             // CheckSum calculation
184                             str_Length:=INT_TO_STRING(LEN(CONCAT(str_CommandID,',',Script_Command)));

                        //Length for DATA command
185                             str_Checksum_Calc :=
    CONCAT(CONCAT(Header,',',str_Length,',',str_CommandID),CONCAT(',',Script_Command),','); //Checksum
    includes: Header, Length for DATA command, str_CommandID and Script_Command
186
187
        Checksum_Length:=ToAryByte(str_Checksum_Calc,_eBYTE_ORDER#_LOW_HIGH,Send_Checksum[0]);
188
189                             IF Checksum_Length>0 THEN
190                                     FB_step:=50;
191                             ELSE
192                                     ToError_Sts := TRUE;
193                                     ToErrorID:=16#40;
194                                     ToErrorDescrip:='Checksum length not valid';
195                             END_IF;
196
197
198
199         50: (*  Frame buildingto send command to TMflow *)
200
201                             FOR i:=1 TO LEN(str_Checksum_Calc) BY 1 DO
202                                     Send_Checksum[0]:=Send_Checksum[0] XOR Send_Checksum[i];
203                             END_FOR;
204
205                             str_Checksum:=BYTE_TO_STRING(Send_Checksum[0]);
206                             str_SendFrame :=
    CONCAT(CONCAT('$$',Header,',',str_Length,','),CONCAT(str_CommandID,',',Script_Command),',*',str_Checksum,'$R$L');
207
208                             Length:=LEN(str_SendFrame);
209                             Long:=ToAryByte(str_SendFrame, _eBYTE_ORDER#_LOW_HIGH,TCP_Send_Data[0]);

210
211                             IF Long>0 THEN
212                                     FB_step:=60;
213                             ELSE
214                                     ToError_Sts := TRUE;
215                                     ToErrorID:=16#50;
216                                     ToErrorDescrip:='Final frame building is error end';
217                             END_IF;
218
219
220         60: (*      Send command                    *)
221
222                             TCP_Send_Size:=LEN(str_SendFrame);
223                             ToAryByte(In:=str_SendFrame, Order:=_eBYTE_ORDER#_LOW_HIGH,
    AryOut:=TCP_Send_Data[0]);
224
225                             TCP_Send_Exe        :=TRUE;
226
227                             IF TCP_Send.Done THEN
228                                     TCP_Send_Exe:=FALSE;
229                                     //FB_step:=70;
230                                     FB_step:=90;                                    //This FB does not checks the
    return confirmation data from robot controller.
```

```
231                             END_IF;
232
233                             IF TCP_Send.Error THEN
234                                     TCP_Send_Exe:=FALSE;
235                                     ToError_Sts := TRUE;
236                                     ToErrorID:=16#60;
237                                     ToErrorDescrip:='TCP send error';
238                             END_IF;
239
240
241     70: (*      Request receiving data                  *)
242
243                             TCP_Rcv_TimeOut:=0;                                     //0: No timeouts
244                             TCP_Rcv_Size:=256;                                      //Set number of bytes to read from
    the receive buffer
245                             StringOfReceivedData:='';               //Clear the variable where Receive data array is compiled
246
247                             TCP_Rcv_Exe :=TRUE;
248
249                             IF TCP_Rcv.Done THEN
250                                     StringOfReceivedData:=AryToString(TCP_Rcv_Data[0],TCP_Rcv_Size);
        //Converts a maximum of 1985 BYTE array to a text string (starting from index [0])
251                                     TCP_Rcv_Exe:=FALSE;
252                                     FB_step:=80;
253                             END_IF;
254
255                             IF TCP_Rcv.Error THEN
256                                     TCP_Rcv_Exe:=FALSE;
257                                     ToError_Sts := TRUE;
258                                     ToErrorID:=16#70;
259                                     ToErrorDescrip:='TCP receive error';
260                             END_IF;
261
262
263     80: (*      Check acknowledgement Command accepted                  *)
264
265                     IF FIND(StringOfReceivedData,'OK') <> 0 THEN                    //Command accepted
266                             FB_step:=90;
                                                                    //Done FB when motion is finished (no blending)
267                     ELSIF FIND(StringOfReceivedData,'ERROR') <> 0 THEN
268                             ToError_Sts := TRUE;
269                             ToErrorID:=16#80;
270                             ToErrorDescrip:=' Command rejected';
271                     END_IF;
272
273
274     90: (*                  End Execution           *)
275                     ToDone_Sts:=TRUE;
276
277     END_CASE ;
278
279 END_IF;
280
281
282
283
284 (* Function Blocks                                          *)
285 (* ---------------------------------------------------------------------- *)
286
287
288 TCP_Clear_Buffer(
289     Execute:=TCP_Clear_Buffer_Exe,
290     Socket:=CBT.Socket
291     //Done=>, Busy=>, Error=>, ErrorID=>
292     );
293
294 TCP_Send(
295     Execute:=TCP_Send_Exe,
296     Socket:=CBT.Socket,
297     SendDat:=TCP_Send_Data[0],
298     Size:=TCP_Send_Size
```

```
299         //Done=>, Busy=>, Error=>, ErrorID=>
300         );
301
302     TCP_Rcv(
303         Execute:=TCP_Rcv_Exe,
304         Socket:=CBT.Socket,
305         TimeOut:=TCP_Rcv_TimeOut,
306         Size:=TCP_Rcv_Size,
307         RcvDat:=TCP_Rcv_Data[0],
308         //Done=>, Busy=>, Error=>, ErrorID=>,
309         RcvSize=>TCP_Rcv_RcvSize);
310
311
312
```

Annex B: Program example

B1. Pick and Place sequence

```
(* ---------------------------------------------------------------------------
Prg Name:               PnP_ExampleV1.44
Date:                   June 2021
Author:                 Ruben Sanchez Boli
Description:            Pick And Place cycle
--------------------------------------------------------------------------- *)


// Detect rising flag on Execute input
R_TRIG_Execute(Clk:=PnP_Example_Execute, Q=>flagExecute);

//Sequence initialization
IF flagExecute THEN
    Prg_Step:=1;
END_IF;

IF NOT(PnP_Example_Execute) THEN
    MoveExecute:=FALSE;
    Prg_Step:=0;
END_IF;

IF PnP_Example_Execute THEN
    CASE Prg_Step OF

            0:
                Prg_Step:=0;
                MoveExecute:=FALSE;

            1:
                MoveExecute:=FALSE;
                Timer_Enable:=FALSE;
                Prg_Step:=10;

            10:
                MoveParameters:=PnP_MoveParameters[0];
                Prg_Step:=11;

            11:
                MoveExecute:=TRUE;
                IF MoveDone THEN
                    MoveExecute:=FALSE;
                    Prg_Step:=12;
                END_IF;


            12:
                MoveParameters:=PnP_MoveParameters[1];
                Prg_Step:=13;

            13:
                MoveExecute:=TRUE;
                IF MoveDone THEN
                    MoveExecute:=FALSE;
                    Prg_Step:=14;
                END_IF;


            14:
                MoveParameters:=PnP_MoveParameters[2];
                Prg_Step:=15;

            15:
                MoveExecute:=TRUE;
```

```
64                              IF MoveDone THEN
65                                      MoveExecute:=FALSE;
66                                      Prg_Step:=16;
67                              END_IF;
68
69                      16:
70                              Timer_Enable:=TRUE;
71                              IF Timer_1.Q THEN
72                                      Timer_Enable:=FALSE;
73                                      Prg_Step:=20;
74                              END_IF;
75
76
77      (*-------------------PLACE----------------*)
78
79                      20:
80                              MoveParameters:=PnP_MoveParameters[3];
81                              Prg_Step:=21;
82
83                      21:
84                              MoveExecute:=TRUE;
85                              IF MoveDone THEN
86                                      MoveExecute:=FALSE;
87                                      Prg_Step:=22;
88                              END_IF;
89
90
91                      22:
92                              MoveParameters:=PnP_MoveParameters[4];
93                              Prg_Step:=23;
94
95                      23:
96                              MoveExecute:=TRUE;
97                              IF MoveDone THEN
98                                      MoveExecute:=FALSE;
99                                      Prg_Step:=24;
100                             END_IF;
101
102
103                     24:
104                             MoveParameters:=PnP_MoveParameters[5];
105                             Prg_Step:=25;
106
107                     25:
108                             MoveExecute:=TRUE;
109                             IF MoveDone THEN
110                                     MoveExecute:=FALSE;
111                                     Prg_Step:=26;
112                             END_IF;
113
114                     26:
115                             Timer_Enable:=TRUE;
116                             IF Timer_1.Q THEN
117                                     Timer_Enable:=FALSE;
118                                     Prg_Step:=10;
119                             END_IF;
120
121             ELSE
122                                     Prg_Step:=0;
123
124             END_CASE;
125     END_IF;
126
127
128
129     (* Function Bolcks                                              *)
130     (* ------------------------------------------------------------ *)
131
132     IF PnP_Example_Execute or PnP_MoveBusy or PnP_MoveDone THEN
133
134             PnP_MoveDone := MoveDone;
```

```
135         PnP_Move_CmdID_Ack := Move_CmdID_Ack;
136         PnP_MoveBusy := MoveBusy;
137         PnP_MoveError := MoveError;
138         PnP_MoveErrorDescrip := MoveErrorDescrip;
139
140         CBT_MoveAbsolute_Instance(
141                     CBT         :=Cobot,
142                     Execute     :=MoveExecute,
143                     Parameters:=MoveParameters,
144                     Done        =>MoveDone,
145                     CmdID_Ack=>Move_CmdID_Ack,
146                     Busy        =>MoveBusy,
147                     Error       =>MoveError,
148                     ErrorID     =>MoveErrorID,
149                     ErrorDescrip=>MoveErrorDescrip);
150
151
152         Timer_1(In:=Timer_Enable, PT:=T#2000ms, Q=>Timer_Done);
153
154     END_IF;
```