MotoLogix
Rockwell Studio 5000
PLC Manual

Upon receipt of the product and prior to initial operation, read these instructions thoroughly, and retain for future reference.

MOTOMAN INSTRUCTIONS
MOTOLOGIX PLC MANUAL
DX200 INSTRUCTIONS
DX200 OPERATOR’S MANUAL (for each purpose)
DX200 MAINTENANCE MANUAL

Part Number: 178631-1CD
Revision: 0
# Table of Contents

1 General .......................................................................................................................... 1-1  
  1.1 Safety Standard for Industrial Robots and Robot Systems .................................................. 1-1  
  1.2 Notes for Safe Operation .................................................................................................. 1-2  
  1.3 About This Manual ......................................................................................................... 1-3  
  1.4 Safety ........................................................................................................................... 1-4  
    1.4.1 Emergency Stop ........................................................................................................ 1-4  
  1.5 Registered Trademarks ................................................................................................... 1-6  
  1.6 Abbreviations ................................................................................................................ 1-6  
  1.7 Customer Support Information ......................................................................................... 1-7  

2 Requirements .................................................................................................................. 2-1  

3 System Setup .................................................................................................................. 3-1  
  3.1 PLC Settings .................................................................................................................. 3-1  
    3.1.1 Configuration ............................................................................................................ 3-1  
    3.1.2 IO Mapping ............................................................................................................... 3-3  
    3.1.3 Task Class Settings ................................................................................................. 3-3  

4 Software .......................................................................................................................... 4-1  
  4.1 Software Package .......................................................................................................... 4-1  

5 MotoLogix Examples ......................................................................................................... 5-1  
  5.1 Introduction .................................................................................................................... 5-1  
    5.1.1 Equipment ................................................................................................................ 5-1  
    5.1.2 YASKAWA Extras Add-On Instructions .................................................................. 5-2  
  5.2 General information ...................................................................................................... 5-3  
    5.2.1 InitOnce .................................................................................................................... 5-3  
    5.2.2 IO Mapping ............................................................................................................. 5-3  
    5.2.3 State Machine: Preparing and Starting .................................................................. 5-3  
    5.2.4 State Machine: Stopping ....................................................................................... 5-5  
    5.2.5 Task Scheduling ..................................................................................................... 5-6  
    5.2.6 Global Variables ................................................................................................. 5-6  
  5.3 Example: General .......................................................................................................... 5-7  
    5.3.1 Description ............................................................................................................... 5-7  
    5.3.2 Communication ..................................................................................................... 5-7  

E1105000011GB
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.3.3</td>
<td>ErrorHandling</td>
<td>5-8</td>
</tr>
<tr>
<td>5.3.4</td>
<td>CurrentPosition</td>
<td>5-10</td>
</tr>
<tr>
<td>5.4</td>
<td>Example: 1_AOIcalls</td>
<td>5-12</td>
</tr>
<tr>
<td>5.5.1</td>
<td>Description</td>
<td>5-13</td>
</tr>
<tr>
<td>5.5</td>
<td>Example: 2_Jog</td>
<td>5-14</td>
</tr>
<tr>
<td>5.5.1</td>
<td>Description</td>
<td>5-14</td>
</tr>
<tr>
<td>5.5.2</td>
<td>Button Panel Usage</td>
<td>5-15</td>
</tr>
<tr>
<td>5.5.3</td>
<td>Motion</td>
<td>5-15</td>
</tr>
<tr>
<td>5.5.3.1</td>
<td>Jog Directions</td>
<td>5-15</td>
</tr>
<tr>
<td>5.5.3.2</td>
<td>Coordinate System</td>
<td>5-16</td>
</tr>
<tr>
<td>5.5.3.3</td>
<td>State Machine</td>
<td>5-17</td>
</tr>
<tr>
<td>5.6</td>
<td>Example: 3_PosTable</td>
<td>5-18</td>
</tr>
<tr>
<td>5.6.1</td>
<td>Description</td>
<td>5-18</td>
</tr>
<tr>
<td>5.6.2</td>
<td>Button Panel Usage</td>
<td>5-19</td>
</tr>
<tr>
<td>5.6.3</td>
<td>Motion</td>
<td>5-19</td>
</tr>
<tr>
<td>5.6.3.1</td>
<td>Position Tracking</td>
<td>5-19</td>
</tr>
<tr>
<td>5.6.3.2</td>
<td>Index</td>
<td>5-19</td>
</tr>
<tr>
<td>5.6.3.3</td>
<td>LoadIndex</td>
<td>5-20</td>
</tr>
<tr>
<td>5.6.3.4</td>
<td>Queuing Amount (QA)</td>
<td>5-20</td>
</tr>
<tr>
<td>5.6.3.5</td>
<td>State Machine</td>
<td>5-20</td>
</tr>
<tr>
<td>5.6.3.6</td>
<td>AOI Mapping</td>
<td>5-22</td>
</tr>
<tr>
<td>5.6.3.7</td>
<td>AOI Status Mapping</td>
<td>5-23</td>
</tr>
<tr>
<td>5.6.4</td>
<td>GenPosData_circle</td>
<td>5-24</td>
</tr>
<tr>
<td>5.7</td>
<td>Example: 4_ConveyorTracking</td>
<td>5-25</td>
</tr>
<tr>
<td>5.7.1</td>
<td>Description</td>
<td>5-25</td>
</tr>
<tr>
<td>5.7.2</td>
<td>Button Panel Usage</td>
<td>5-26</td>
</tr>
<tr>
<td>5.7.3</td>
<td>Encoder Card Wiring</td>
<td>5-27</td>
</tr>
<tr>
<td>5.7.4</td>
<td>ConveyorCalc</td>
<td>5-28</td>
</tr>
<tr>
<td>5.7.4.1</td>
<td>Encoder Handling</td>
<td>5-28</td>
</tr>
<tr>
<td>5.7.4.2</td>
<td>Object Queue</td>
<td>5-28</td>
</tr>
<tr>
<td>5.7.4.3</td>
<td>Size</td>
<td>5-29</td>
</tr>
<tr>
<td>5.7.4.4</td>
<td>Conveyor Position</td>
<td>5-29</td>
</tr>
<tr>
<td>5.7.4.5</td>
<td>Add Object</td>
<td>5-29</td>
</tr>
<tr>
<td>5.7.4.6</td>
<td>Remove Object</td>
<td>5-30</td>
</tr>
<tr>
<td>5.7.5</td>
<td>Motion</td>
<td>5-31</td>
</tr>
<tr>
<td>5.7.5.1</td>
<td>State Machine</td>
<td>5-31</td>
</tr>
</tbody>
</table>

6 System Performance | 6-1

6.1 Results | 6-1

6.2 Memory Usage | 6-1
1 General

1.1 Safety Standard for Industrial Robots and Robot Systems

We suggest that you obtain and review a copy of the ANSI/RIA National Safety Standard for Industrial Robots and Robot Systems (ANSI/RIA R15.06-2012). You can obtain this document from the Robotic Industries Association (RIA) at the following address:

Robotic Industries Association
900 Victors Way
P.O. Box 3724
Ann Arbor, Michigan 48106
TEL: (734) 994-6088
FAX: (734) 994-3338
www.roboticsonline.com

Ultimately, well-trained personnel are the best safeguard against accidents and damage that can result from improper operation of the equipment. The customer is responsible for providing adequately trained personnel to operate, program, and maintain the equipment. NEVER ALLOW UNTRAINED PERSONNEL TO OPERATE, PROGRAM, OR REPAIR THE EQUIPMENT!

We recommend approved YASKAWA training courses for all personnel involved with the operation, programming, or repair of the equipment.

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to part 15 of the FCC rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications.
1.2 Notes for Safe Operation

Read this manual carefully before installation, operation, maintenance, or inspection of the DX200.

In this manual, the Notes for Safe Operation are classified as “DANGER”, “WARNING”, “CAUTION”, “MANDATORY”, or “PROHIBITED”.

- **DANGER**: Indicates an imminent hazardous situation which, if not avoided, could result in death or serious injury to personnel.
- **WARNING**: Indicates a potentially hazardous situation which, if not avoided, could result in death or serious injury to personnel.
- **CAUTION**: Indicates a potentially hazardous situation which, if not avoided, could result in minor or moderate injury to personnel and damage to equipment. It may also be used to alert against unsafe practices.
- **MANDATORY**: Always be sure to follow explicitly the items listed under this heading.
- **PROHIBITED**: Must never be performed.

Even items described as “CAUTION” may result in a serious accident in some situations.

At any rate, be sure to follow these important items.

To ensure safe and efficient operation at all times, be sure to follow all instructions, even if not designated as “DANGER”, “WARNING” and “CAUTION”.

---

MotoLogix Rockwell Studio 5000 PLC

1 General
1.2 Notes for Safe Operation
1.3 About This Manual

• The MotoLogix Function of robot controller from YASKAWA provides an easy way to develop PLC-based robotic solutions. Programming of the MotoLogix is performed entirely using PLC through the use of a set of Add-On Instructions (Rois) for robot motion and configuration. This manual describes just the Rockwell platform specific functionality. To get more detailed information about general MotoLogix system architecture and programming, refer to the common manual “Software and Operating Instructions, MotoLogix Common Section”.

• To ensure correct and safe operation, read the robot controller operating instructions (Basic Information, Installation and Wiring and System Setup).

• In order to illustrate details clearly, some drawings are shown with the protective covers or shields removed. All protective covers and shields must be mounted before the robot is operated.

• The drawings and figures in this manual are representative illustrations. They may differ from the product delivered.

• YASKAWA reserves the right to make technical changes. These changes may include product improvements, modifications or changes in specifications.

• If your copy of the operating and maintenance instructions are damaged or lost, please contact the local YASKAWA branch office to order a new copy. The official branch offices are listed on the last page. Please mention the part number with your order.

• YASKAWA is not responsible for damage caused due to unauthorized modification of the system. If any impermissible modifications are made to the system and to the robot, all warranty and liability claims will be voided.
1.4 Safety

1.4.1 Emergency Stop

Fig. 1-1: Programming Pendant

WARNING

Death or injury because of danger of crushing

If the emergency stop button does not function properly, the robot cannot be stopped in the event of an emergency.

- The robot should not be used if the emergency stop button does not function.
- Before operating the robot check the function of the emergency stop button. The SERVO power has to immediately go off once the emergency stop button on the programming pendant has been pressed (see Fig. 1-3 “Emergency Stop Button”).
- When the SERVO power is turned OFF, the SERVO ON LED on the programming pendant goes off (see Fig. 1-2 “LED SERVO ON”).

Fig. 1-2: LED SERVO ON

SERVO ON

Fig. 1-3: Emergency Stop Button
WARNING

Death or injury because of danger of crushing
Before you release the emergency stop button (see Fig. 1-4 “Release of Emergency Stop Button by Turning”) note the following:

- Make sure that there is no one within the maximum working range of the robot.
- Clear the cell of all items which could collide with the robot.
- Now you can switch ON the SERVO power by pressing the enable switch on the programming pendant.

Fig. 1-4: Release of Emergency Stop Button by Turning

WARNING

Death or injury because of danger of crushing
If anyone enters the working area of the robot during operation or any problems occur, always press the emergency stop button immediately. The emergency stop button is located on the programming pendant (see Fig. 1-1 “Programming Pendant”).

Observe the following precautions when performing teaching operations within the robot’s working range:

- View the Robot from the front whenever possible.
- Always follow the prescribed operating procedure (see the instructions on robot control as well as the operating instructions on “Handling” or “Universal Application”).
- An area must be left clear so that the operator can retreat to it in case of emergency.

The following inspection procedures must be performed prior to teaching the robot. If problems are found, correct them immediately, and be sure that all other necessary measures have been performed.

- Check for problems in robot movement.
- Check the connectors for tight fit and all cables for damage.
- Hang the programming pendant back on the robot control after use.
- Make sure that the key for the key switch (Teach/Automatic) of the programming pendant is kept by a skilled person who has been specially trained.
- The key may be inserted in the key switch of the programming pendant only during teach operation; after the teach operation it must be immediately removed and kept in a safe place.
1.5 Registered Trademarks

In this manual, names of companies, corporations, or products are trademarks, registered trademarks, or brand names for each company or corporation. The indications of ® and ™ are omitted.

1.6 Abbreviations

*Table 1-1: Abbreviations*

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Meaning</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB</td>
<td>Function Block</td>
<td>Rockwell calls it Add-On Instruction</td>
</tr>
<tr>
<td>AOI</td>
<td>Add-On Instruction</td>
<td>Rockwell name for a Function Block</td>
</tr>
<tr>
<td>FC</td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>Inputs and Outputs</td>
<td></td>
</tr>
<tr>
<td>DI</td>
<td>Digital Input</td>
<td></td>
</tr>
<tr>
<td>DO</td>
<td>Digital Output</td>
<td></td>
</tr>
</tbody>
</table>
1.7 Customer Support Information

If you need assistance with any aspect of your MotoLogix Rockwell Studio 5000 PLC system, please contact YASKAWA Customer Support at the following 24-hour telephone number:

(937) 847-3200

For routine technical inquiries, you can also contact YASKAWA Customer Support at the following e-mail address:

technology@motoman.com

When using e-mail to contact YASKAWA Customer Support, please provide a detailed description of your issue, along with complete contact information. Please allow approximately 24 to 36 hours for a response to your inquiry.

Please use e-mail for routine inquiries only. If you have an urgent or emergency need for service, replacement parts, or information, you must contact YASKAWA Customer Support at the telephone number shown above.

Please have the following information ready before you call Customer Support:

- **System**: MotoLogix Rockwell Studio 5000 PLC
- **Primary Application**: ___________________________
- **Controller**: DX200
- **Software Version**: Access this information on the Programming Pendant’s LCD display screen by selecting {MAIN MENU} - {SYSTEM INFO} - {VERSION}
- **Robot Serial Number**: Located on the robot data plate
- **Robot Sales Order Number**: Located on the DX200 controller data plate
## Table 2-1: Hardware and Software Requirements

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Rockwell ControlLogix, CompactLogix or GuardLogix controller, Ethernet/IP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1Mb of available PLC memory</td>
</tr>
<tr>
<td></td>
<td>YASKAWA DX200 robot controller with following options:</td>
</tr>
<tr>
<td></td>
<td>• MotoLogix Runtime</td>
</tr>
<tr>
<td></td>
<td>• Ethernet/IP</td>
</tr>
<tr>
<td>Software</td>
<td>Rockwell Studio v21</td>
</tr>
</tbody>
</table>
3 System Setup

The picture below shows a possible system layout. One MotoLogix system can handle up to four robots. A single PLC can control multiple MotoLogix systems. This is only limited by the PLC and fieldbus capacity. The dashed lines are optional devices or connections.

Fig. 3-1: System Layout

3.1 PLC Settings

This chapter explains how to setup your PLC project for MotoLogix. These steps can be skipped if you use the supplied MotoLogix PLC project from the software package. This project is already configured for one MotoLogix system.

3.1.1 Configuration

1. Add a new module to the Ethernet network.

Fig. 3-2: New Module
2. Select the Allen-Bradley Generic Ethernet Module.

*Fig. 3-3: Generic Ethernet Module*

3. Enter these settings for the new module.

*Fig. 3-4: Setting for New Module*
3.1.2 IO Mapping

Next map the fieldbus data to the variables. In this example two global tags for the fieldbus data were created.

*Fig. 3-5: Global Variables for Fieldbus Data*

<table>
<thead>
<tr>
<th>MLX_0_Input</th>
<th>MLX_0_Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>MotoLogix_01</td>
<td>MotoLogix_00</td>
</tr>
<tr>
<td>AB-ETHERNET_MODULE_SINT_48Bytes:0</td>
<td>AB-ETHERNET_MODULE_SINT_48Bytes:0</td>
</tr>
<tr>
<td>data from MotoLogix Runtime</td>
<td>data to MotoLogix Runtime</td>
</tr>
</tbody>
</table>

3.1.3 Task Class Settings

The MotoLogix Runtime of the robot controller has a cycle of 4 ms. It is recommended to use a task period <4 ms to ensure the shortest processing times for a MotoLogix application. See section 6.1 "Results" on page 6-1.
4 Software

4.1 Software Package

The package contains the following files:

• MotoLogix_Vx_x_x.zip

■ MotoLogix PLC Project

The supplied MotoLogix PLC project contains the hardware configuration, Add-On Instructions and Examples (for one MotoLogix system) and IO mapped global variables.

Use this project as start off point.

1. Extract MotoLogix_Vx_x_x.zip
2. Open the project.
3. Compile to verify for errors.
5 MotoLogix Examples

The following chapters show a few sample programs to demonstrate the use of the MotoLogix library. These examples can help with the understanding of how certain motion applications could be programmed.

The difficulty of the examples increases with the numbering and for persons who are new to MotoLogix it is advised to start from the beginning.

Fig. 5-1: MotoLogix Examples

5.1 Introduction

The example programs are partly written in Ladder and partly in Structured Text (ST) and have been designed and tested by YASKAWA.

5.1.1 Equipment

• Allen-Bradley 1769-L33ER CompactLogix PLC with additional cards:
  – 1769-IQ16 DI card
  – 1769-OB16 DO card
  – 1769-HSC Encoder card

• MH12 robot mounted on a 300 mm high base. No Tool attached.

• Conveyor
  – driven by AC motor + VFD
  – incremental encoder and photo eye connected to encoder card
5 MotoLogix Examples

5.1 Introduction

• Button panel with
  – four (illuminated) pushbuttons (Din[0-3], Dout[0-3])
  – four selector switches (Din[4-7])

The functionality of the button panel varies and is documented for each example.

Fig. 5-2: Button Panel

5.1.2 YASKAWA Extras Add-On Instructions

The software package contains some YASKAWA Extras Add-On Instructions. These Add-On Instructions have the prefix “Ext” and are not tied to MotoLogix. Some of the example programs make use of these.

Although the functionality has been tested we must state the following about the YASKAWA Extras Add-On Instructions:

• These are not officially supported by YASKAWA
• There are no help documents available
• They are not knowhow protected

Fig. 5-3: Add-On Instructions
5.2 General information

Since the example programs have a similar structure this section explains the topics which are applicable for all of them.

5.2.1 InitOnce

The data used by the examples (such as Userframes, Positions, Tools) is initialized locally for each example in the InitOnce (at startup). This data might need to change to fit the robot for a particular application.

5.2.2 IO Mapping

Most examples make use of the button panel. The mapping of these DI or DO signals is handled locally for each example in the main routine.

Fig. 5-4: DI Mapping Example

5.2.3 State Machine: Preparing and Starting

When diOnOff is switched on the state machine will prepare and start the MotoLogix system. This also includes setting the Tool and UserFrame.
Fig. 5-5: Prepare and Start Logic Example
### 5.2.4 State Machine: Stopping

When **diOnOff** is switched off the state machine will stop the MotoLogix system.

*Fig. 5-6: Stop Logic Example*
5.2.5 Task Scheduling

A two millisecond task class is used for the examples.

*Fig. 5-7: Task Scheduling*

```
NOTE

Example 2, 3, and 4 cannot be active at the same time. Always make sure that no more than one is scheduled.
```

5.2.6 Global Variables

The amount of global variables is kept to a minimum.

*Fig. 5-8: Global Variables*
5.3 Example: General

Fig. 5-9: Program: General

5.3.1 Description

This program has following functionality:

- Communication with the MotoLogix Runtime
- Reading the MotoLogix Runtime errors or alarms
- Reading current position data for the various coordinate systems of all robots

It is required to have this program always scheduled since all other example programs depend on the communication with the MotoLogix Runtime.

5.3.2 Communication

For communication we need one MLxCommunication call for each MotoLogix system. Each YASKAWA robot controller with a fieldbus interface and activated MotoLogix Runtime counts as a MotoLogix system.

Fig. 5-10: Communication
5.3.3 ErrorHandling

The MotoLogix Runtime can have multiple errors or alarms (here called: errors). This program uses a state machine for reading these errors. The amount of errors is checked and whenever new errors arise it will read all errors one after another using MLxGetErrorDetail. The fetched error texts are stored in ErrorDetail[0..9] OF MLxErrorDetail. When the amount of errors becomes 0 this array is cleared.

Fig. 5-11: ErrorHandling 01
Fig. 5-12: ErrorHandling 02

The jump label can be used if only one state change per PLC scan is allowed. It results in the same behaviour as a CASE...OF statement in ST.
5.3.4 CurrentPosition

For reading actual position data MotoLogix cyclically sends:

- Each axis position for each robot
- TCP position for one robot in one coordinate system

The latter makes use of the following variables:

- MLX[].HMIFeedbackData
- MLX[].HMIFeedbackConfiguration

We can specify the robot number and coordinate system in HMIFeedbackConfiguration and from then on HMIFeedbackData will contain the desired TCP position data.

The axis positions are available in the


By using the MLxxCopyAxisDataToReal Add-On Instruction we can copy all axes of one robot to an array.

This program uses a state machine for subsequently reading the actual Axis, World or UserFrame position data for four robots. This is done continuously, resulting in fast updates for all position data. It is not affecting the overall performance of MotoLogix since it doesn’t use the same communication resource as the Add-On Instructions.

Fig. 5-13: CurrentPosition 01
As we do not use MLx Add-On Instructions in this example we cannot use the usual handshaking mechanism (for example Sts_DN). Therefore we will detect new MotoLogix data by observing the SynchCounter (general communication handshake, also used for watchdog). After specifying a different robot or coordinate system we always wait two successful communication cycles to make sure MotoLogix sends the specified actual data.
5.4 Example: _AOIcalls

Fig. 5-16: Program: _AOIcalls

<table>
<thead>
<tr>
<th>Description</th>
<th>Example 1: Call of each individ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>Inhibited</td>
</tr>
<tr>
<td>Number of Routines</td>
<td>3</td>
</tr>
<tr>
<td>Main Routine</td>
<td>Main</td>
</tr>
<tr>
<td>Fault Routine</td>
<td></td>
</tr>
<tr>
<td>Max Scan</td>
<td></td>
</tr>
<tr>
<td>Last Scan</td>
<td></td>
</tr>
</tbody>
</table>
5.4.1 Description

The program contains a variable declaration and call for each MotoLogix Add-On Instruction. Variables are declared as arrays so you can easily change it to the amount of instances you need.

Practice with the different AOIs by toggling the _Enable bits to get familiar with the functionality of each Add-On Instruction. This program can also be useful as a source of “AOI call snippets” when you program your own application.

Fig. 5-17: Variables for MotoLogix AOIs

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB_Abort</td>
<td>MLxAbort[1]</td>
</tr>
<tr>
<td>FB_Abort_Enable</td>
<td>BOOL</td>
</tr>
<tr>
<td>FB_BvarsGet</td>
<td>MLxBvarsGet[1]</td>
</tr>
<tr>
<td>FB_BvarsGet_Enable</td>
<td>BOOL</td>
</tr>
<tr>
<td>FB_BvarsSet</td>
<td>MLxBvarsSet[1]</td>
</tr>
<tr>
<td>FB_BvarsSet_Enable</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

Fig. 5-18: Example of an AOI Call
5.5 Example: 2_Jog

Fig. 5-19: Program: Jog

5.5.1 Description

This program has following functionality:

• Preparing and starting the MotoLogix system
• Jogging the robot TCP in the various coordinate systems (for example World, UserFrame, Tool)
• Jogging the robot axis (for example S-axis, L-axis, U-axis)
• Stopping the MotoLogix system

A state machine is used for handling these tasks. The defined states are:

```
CASE state OF
  (*************************** state 0 - idle ***************************)
  (*************************** state 5 - reset system ***************************)
  (*************************** state 6 - set Tool ***************************)
  (*************************** state 7 - set UserFrame ***************************)
  (*************************** state 8 - enable system ***************************)
  (*************************** state 10 - ready for jog ***************************)
  (*************************** state 11 - jog TCP ***************************)
  (*************************** state 13 - jog axes ***************************)
  (*************************** state 21 - abort system ***************************)
  (*************************** state 22 - reset system ***************************)
END_CASE;
```
5.5.2 Button Panel Usage

1. **diJogPos[0]** - jog first axis: POS
   doStatus - application active
   Flashing: standby (communication OK)

2. **diJogNeg[0]** - jog first axis: NEG

3. **diJogPos[1]** - jog second axis: POS

4. **diJogNeg[1]** - jog second axis: NEG

5. **diOnOff** - activate application

6. **diCoordFrameSelectBit1** - CoordFrame selection (bit 1)

7. **diCoordFrameSelectBit0** - CoordFrame selection (bit 0)

<table>
<thead>
<tr>
<th>CoordFrame</th>
<th>diCoordFrame SelectBit1</th>
<th>diCoordFrame SelectBit0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - World</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1 - Tool</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2 - User</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3 - Axes</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

5.5.3 Motion

5.5.3.1 Jog Directions

For Jogging the robot in this example **MLxRobotJogTCP** and **MLxRobotJogAxes** are used. Both of these use **Directions[0..7]** OF **DINT** as parameter for setting the Jog direction per axis. These directions are continuously set based on the status of the **JogNeg** and **JogPos** buttons.

```java
// translate jog buttons to jog directions

// set jog direction per axis; pos: 1, neg: -1
JogPressed := 0;
SIZE(Directions, 0, arraysize);
FOR i:=0 TO arraysize-1 DC
    Directions[i] := 0;
    IF diJogPos[i] AND NOT diJogNeg[i] THEN
        JogPressed := 1;
    END_IF;
    IF diJogNeg[i] AND NOT diJogPos[i] THEN
        JogPressed := 1;
    END_IF;
    IF diJogPos[i] AND diJogNeg[i] THEN
        JogPressed := 1;
    END_IF;
END_FOR;
END_VAR;
```
5.5.3.2 Coordinate System

The selection of the coordinate system is done by two selector switches.

- Selector switches for choosing CoordFrame: 0 = World, 1 = Tool, 2 = User, 3 = Axes

```plaintext
<table>
<thead>
<tr>
<th>CoordFrame</th>
<th>0 e</th>
</tr>
</thead>
</table>

- Source A

```plaintext
<table>
<thead>
<tr>
<th>CoordFrame</th>
<th>0 e</th>
</tr>
</thead>
</table>
```

- Source B

```plaintext
<table>
<thead>
<tr>
<th>CoordFrame</th>
<th>0 e</th>
</tr>
</thead>
</table>
```

- Dest

```plaintext
<table>
<thead>
<tr>
<th>CoordFrame</th>
<th>0 e</th>
</tr>
</thead>
</table>
```
After preparing and starting, the system is ready for jogging (state 10). If a jog button is pressed it will enter the state for TCP or axis jogging (based on the selected coordinate system). As described in the manual, the Jog AOIs need to be called intermittent. For this reason the state machine will return to state 10 as soon as the jog instruction returns the Sts_DN. In state 10 it will wait for a short time (Idle time) before proceeding to the next state. During this time other motion instructions can be processed.
5.6 Example: 3_PosTable

Fig. 5-20: Program: PosTable

5.6.1 Description

This program has following functionality:

• Preparing and starting the MotoLogix system

• Positioning the robot to (TCP) positions stored in a table (array)
  – Linear and non-linear (ptp) moves
  – Attributes (Speed, BlendFactor, Linear/ptp) adjustable for each single position
  – Flexible: to adjust the amount of positions only the array size need to be changed
  – Robust: keeps track of the current and preloaded motions (easy to continue after unexpected stop or error)
  – Selector switches for:
    Waiting after table has finished (that is end of table or after the last enabled position)
    Waiting after each single move
    Enabling preloading the next two positions (queuing) for smooth motion

• Generating position data by using mathematic formula

• Stopping the MotoLogix system
A state machine is used for handling these tasks. The defined states are:

```
CASE state OF
  (%-------------------------- state 0 - idle --------------------------)
  (%-------------------------- state 5 - reset system --------------------------)
  (%-------------------------- state 6 - set Tool --------------------------)
  (%-------------------------- state 7 - set UserFrame --------------------------)
  (%-------------------------- state 8 - enable system --------------------------)
  (%-------------------------- state 11 - move to pos n / queue pos n+1, n+2 --------------------------)
  (%-------------------------- state 12 - move to pos n+1 / queue pos n+2, n --------------------------)
  (%-------------------------- state 13 - move to pos n+2 / queue pos n, n+1 --------------------------)
  (%-------------------------- state 19 - waiting at end of table --------------------------)
  (%-------------------------- state 21 - abort system --------------------------)
  (%-------------------------- state 22 - reset system --------------------------)
END_CASE;
```

### 5.6.2 Button Panel Usage

1. **doStatus** - application active
   - Flashing: standby (communication OK)
2. **diStep** - step button
   - **doStep** - busy
   - Flashing: waiting for step button
3. **diOnOff** - activate application
4. **diContinuous** - do not wait for step button after each move
5. **diWaitAfterLastEntry** - wait for step button after last move
6. **diUseQueuing** - use queuing (preload next motions)

### 5.6.3 Motion

#### 5.6.3.1 Position Tracking

To keep track of the current and preloaded motions the variables **Index**, **LoadIndex** and **QA** are used.

#### 5.6.3.2 Index

Index is the current motion (array entry number) and will be updated by **UpdateIndex** after the current motion has completed. Before starting the application you can change from which position in the table it should start by setting the value of Index.
5.6.3.3 LoadIndex

LoadIndex is the motion (array entry number) last loaded by a move AOI. It is updated by UpdateLoadIndex on the rising edge of each Enable bit. When queuing is not active (and for the very first move), it will have the same value as Index. When queuing is active, the value will be different from index since it queues (preloads) up to two motions.

5.6.3.4 Queuing Amount (QA)

The variable QA represents the allowed number of motions to be queued. Its value depends on several conditions and is calculated in UpdateIndex. For example QA will be reduced to 0 at the time it reaches the end of the table when WaitAfterLastEntry is active.

5.6.3.5 State Machine

After preparing and starting, the system will start moving to the positions stored in the myPosTable using state 11, 12 and 13. Depending on the state of WaitAfterLastEntry it will wait at the end of the table (state 19) or immediately start over from the beginning.
5 MotoLogix Examples

5.6 Example: 3_PosTable
TransitionFlash, diContinuous and RE_Step are part of the mechanism where the system waits for the user to press Step button to continue. A flashing indicator informs the user that the system is waiting.

### 5.6.3.6 AOI Mapping

The enable signals for the (move) AOIs are not directly written but instead bits 0-2 in Enable (DINT) are used. On the rising edge of each Enable bit the parameter data is loaded from myPosTable.

```cpp
// map parameter data to move FV's
S|D|E|B|A [D] (0-2) Enable
FTP (1, 0-2, [Enable])
IF Enable[i] THEN
   JFB (UpdateLoadIndex);
   UseMoveLinear.[i]; := myPosTable[LoadIndex].MoveLinear;
   := 0;
   MoveFrom[i].RobotNumber;
   CPD[myPosTable[LoadIndex]].TCPPosition[0], MoveFrom[i].TargetPosition.TCPPosition.TCPPosition[0], 0;
   MoveFrom[i].TargetType
   MoveFrom[i].BlendFactor
   MoveFrom[i].BlendType
   IF [SpeedOverride * 5] THEN
   MoveFrom[i].Speed
   ELSE
   MoveFrom[i].Speed
   ENDF;
   MoveFrom[i].UseRotationalSpeed
   IF myPosTable[LoadIndex].SpeedUnits THEN
   MoveFrom[i].SpeedUnits
   ELSE
   MoveFrom[i].SpeedUnits
   ENDF;
   MoveFrom[i].Acceleration
   MoveFrom[i].Deceleration
   ENDF;
END_FPV;
```

With variable SpeedOverride we can override the speed of the myPosTable entry by setting a percentage. 0=original speed, xxx=xxx% of the original speed (there is no check on the speed limit).
The Enable bits will activate the corresponding move AOI, depending on the linear or non-linear setting specified.

5.6.3.7 AOI Status Mapping

The status signals of the move AOIs are also mapped. For this the variable MoveCmd [0..2] OF MoveCmdStatus is used. MoveCmdStatus is a data type containing the various status signals (for example Sts_EN, Sts_PC etc.).

```c
// map statusdata from move FS'
//-----------------------------------------------------------------------
SIZE(MoveCmd, 0, arraysize);
FOR i := 0 TO arraysize DO
  IF UsedMoveLinear[i] THEN
    MoveCmd[i].Sts_EN := FB_RobotMoveLinearAbsolute[i].Sts_EN;
    MoveCmd[i].Sts_PC := FB_RobotMoveLinearAbsolute[i].Sts_PC;
    MoveCmd[i].Sts_AC := FB_RobotMoveLinearAbsolute[i].Sts_AC;
    MoveCmd[i].Sts_IP := FB_RobotMoveLinearAbsolute[i].Sts_IP;
    MoveCmd[i].Sts_ER := FB_RobotMoveLinearAbsolute[i].Sts_ER;
  ELSE
    MoveCmd[i].Sts_EN := FB_RobotMoveAxisAbsolute[i].Sts_EN;
    MoveCmd[i].Sts_PC := FB_RobotMoveAxisAbsolute[i].Sts_PC;
    MoveCmd[i].Sts_A := FB_RobotMoveAxisAbsolute[i].Sts_AC;
    MoveCmd[i].Sts_D := FB_RobotMoveAxisAbsolute[i].Sts_D;
    MoveCmd[i].Sts_ER := FB_RobotMoveAxisAbsolute[i].Sts_ER;
  END_IF;
END_FOR;
```
5.6.4 GenPosData_circle

Using a math formula we can easily fill myPosTable with position data for testing. The routine checks the size of myPosTable and fills it evenly with the calculated positions.

The generated position data forms a circular movement with a constant speed and same distance between the segments. It can be useful for testing the throughput of the system ("how many move commands can it handle per second"). Another use: By setting the size of myPosTable to 4 it will make a square movement which can be used to see the effect of the BlendFactor on the trajectory.

```plaintext
// calculate positiondata: circle (diameter 800mm)
// x(t) = 400 * sin(t), 400 * cos(t)
// the positiondata for tMin to tMax is spread over the arrayentries.
// option 1: endpos is followed by startpos (myPosTable[0..3] results in a square)
// t = (i * (tMax-tMin) / (arraysize-1)) + tMin;
// option 2: start- and endpos are equal (myPosTable[0..4] results in a square)
// t = (i * (tMax-tMin) / (arraysize-1)) + tMin;

myPosTable[1].Enabled := 1;
myPosTable[1].MoveLinear := 1;
myPosTable[1].TCPPosition[0] := 400 * SIN(0); //X
myPosTable[1].TCPPosition[1] := 400 * COS(0); //Y
myPosTable[1].TCPPosition[2] := 800; //Z
myPosTable[1].TCPPosition[3] := 0;
myPosTable[1].TCPPosition[4] := 0;
myPosTable[1].TCPPosition[5] := 0;
myPosTable[1].TCPPosition[6] := 0;
myPosTable[1].TCPPosition[7] := 0;
myPosTable[1].Speed := 12; //speed 150
myPosTable[1].UseRotationalSpeed := 0;
myPosTable[1].SpeedUnits := 0;
myPosTable[1].BlendFactor := -1;
myPosTable[1].UserFrameNumber := 0;
myPosTable[1].ToolNumber := 0;
xmod_pos;
```
5.7 Example: 4_ConveyorTracking

Fig. 5-21: Program: ConveyorTracking

5.7.1 Description

This program has following functionality:

• Preparing and starting the MotoLogix system
• Handling of the encoder card with incremental encoder and fast input for object registration (photo eye)
• Keeping track of multiple objects in an ObjectQueue
• Positioning logic for picking objects from running conveyor
• Stopping the MotoLogix system
A state machine is used for handling these tasks. The defined states are:

```
CASE state GF
  (**-------------------------- state 0 - idle --------------------------)**
  (**-------------------------- state 1 - reset system ------------------------)**
  (**-------------------------- state 2 - set Tool -----------------------------)**
  (**-------------------------- state 7 - set UserFrame ------------------------)**
  (**-------------------------- state 6 - enable system ------------------------)**
  (**-------------------------- state 11 - rest pos (pnp) ---------------------)**
  (**-------------------------- state 12 - rest pos (linear) -------------------)**
  (**-------------------------- state 13 - start sync --------------------------)**
  (**-------------------------- state 14 - synced pick moves -------------------)**
  (**-------------------------- state 15 - stop sync --------------------------)**
  (**-------------------------- state 17 - place moves ------------------------)**
  (**-------------------------- state 21 - abort system -----------------------)**
  (**-------------------------- state 22 - reset system ------------------------)**
END_CASE;
```

### 5.7.2 Button Panel Usage

1. **doStatus** - application active
   - Flashing: standby (communication OK)
2. **doConveyorOn** - run conveyor
3. **diOnOff** - activate application
5.7.3 Encoder Card Wiring

The incremental encoder and photo eye (for object detection) in our setup are wired using following connections:

<table>
<thead>
<tr>
<th>Signal</th>
<th>Wire</th>
<th>Encoder Card</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encoder A</td>
<td>Green</td>
<td>A0-*</td>
</tr>
<tr>
<td>Encoder A1</td>
<td>Yellow</td>
<td>A0+*</td>
</tr>
<tr>
<td>Encoder B</td>
<td>Grey</td>
<td>B0+</td>
</tr>
<tr>
<td>Encoder B1</td>
<td>Pink</td>
<td>B0-</td>
</tr>
<tr>
<td>Photoeye</td>
<td>Blue</td>
<td>Z0+ (Z0- to COM)</td>
</tr>
<tr>
<td>+24V</td>
<td>Brown</td>
<td>+24DC</td>
</tr>
<tr>
<td>0V</td>
<td>White</td>
<td>COM</td>
</tr>
</tbody>
</table>

A0+ and A0- were swapped to fit the direction of our conveyor.

Fig. 5-22: 1769-HSC Pin Out and Wiring
5.7.4 ConveyorCalc

5.7.4.1 Encoder Handling

The ExtHSCLogic Add-On Instruction from the YASKAWA Extras is used for the handling of the High Speed Counter or encoder card. It takes care of reading out the stored encoder value (after capturing) and preparing the encoder card for capturing the next object.

```c
// HSC card handling

// FB_HSCLogic_EnableEncoder
FB_HSCLogic_EnableTrigger

// FB call
EnabHSCLogic(FB_HSCLogic, HSCData(0));
```

Due to the fact that HSCData is a data type we cannot use “alias” to map the Encoder IO directly to the HSCData members. Therefore this is done in the main routine.

Fig. 5-23: Encoder IO Mapping

5.7.4.2 Object Queue

The ObjectQueue keeps track of the objects on the conveyor. For each object a position value ConveyorOffset is stored. This ConveyorOffset is used for calculating the X-offset for the robot during the conveyor tracking movements.
5.7.4.3 Size

The size of the ObjectQueue is set in data type `MLxRobotConveyorData` and can be changed to fit your application.

**Fig. 5-24: ObjectQueue Size**

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueueControl</td>
<td>CONTROL</td>
<td>A control variable tied to the ObjectQueue.</td>
</tr>
<tr>
<td>ObjectQueue</td>
<td>MLxRobotConveyorData</td>
<td>A queue of objects on this conveyor.</td>
</tr>
<tr>
<td>InternalStartPosition</td>
<td>DINT</td>
<td>INTERNAL USE ONLY</td>
</tr>
<tr>
<td>ConvveyorStopStep</td>
<td>DINT</td>
<td>INTERNAL USE ONLY</td>
</tr>
<tr>
<td>ConvveyorMoveQueue</td>
<td>DINT</td>
<td>INTERNAL USE ONLY</td>
</tr>
</tbody>
</table>

5.7.4.4 Conveyor Position

The conveyor position represents the distance (in mm) which the conveyor traveled, measured from the photo eye. The conveyor position is reset every time a first object is added to an empty ObjectQueue.

5.7.4.5 Add Object

Every time a new object is registered by the ExtHSCLogic, it will be added to the ObjectQueue. It means the position will be calculated to mm and stored as a ConveyorOffset value.
5.7.4.6 Remove Object

When an object:
- is successfully picked
- is missed (object already passed the MaxStartPosition when conveyor tracking started)

The oldest object will be removed from the ObjectQueue. This is done when conveyor tracking is stopped by:
- MlxRobotConvSyncStop
- MlxRobotConvSyncStopWithAxisMot
- MlxRobotConvSyncStopWithLinMot

These AOIs set ConveyorStopStep=1 which acts as trigger to remove the object from the ObjectQueue. As part of the handshake we have to set ConveyorStopStep=3 when the object is re-moved.
5.7.5 Motion

5.7.5.1 State Machine

After preparing and starting, the system will move to the defined rest position in state 11. When at rest position the conveyor tracking will be started in state 13. From that moment the system starts checking for objects in the ObjectQueue. If the oldest object in the queue has already passed the MaxStartPosition (Sts_Ol) it is counted as a missed part and conveyor tracking will be stopped (state 16). Else, after passing the ConveyorStartPosition the robot will start the synchronized pick motions in state 14.
The synchronized pick motions are programmed as usual but since conveyor tracking is activated, the X-position of the motions is continuously modified to track the moving object on the conveyor. This is done within the MotoLogix Runtime. The PLC supplies `ConveyorPosition` and `ConveyorOffset` values.

The motion for the pick position must use `BlendFactor=0` and `BlendType=1`. When it reaches its target (`.Sts_PC`) it can pick the object. For simulating a gripper action we use timer `PickTime`. The next motion will lift the object from the conveyor and after that the conveyor tracking is stopped in state 16.

Any place motions could be programmed in state 17. When done, it will either move linear to rest position (state 12) when the `ObjectQueue` is empty or immediately track the next object (state 13).
5 MotoLogix Examples
5.7 Example: 4_ConveyorTracking
6 System Performance

6.1 Results

The following table contains basic performance results measured over 500 samples running MotoLogix on a CompactLogix 1769-L33ER PLC. These results will vary depending on the PLC in use, but should provide a general understanding of the impact and trade-offs.

The measured values reflect the latency between a MotoLogix AOI (MLxRobotMoveLinearAbsolute) being enabled and the command being acknowledged by the MotoLogix Runtime.

Table 6-1: Performance Results

<table>
<thead>
<tr>
<th>Test Description</th>
<th>Average (ms)</th>
<th>Minimal (ms)</th>
<th>Maximal (ms)</th>
<th>Std Dev (ms)</th>
<th>CPU Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.0 ms Ethernet/IP, 2.0 ms Task</td>
<td>10.523</td>
<td>7.705</td>
<td>17.838</td>
<td>2.001</td>
<td>20%</td>
</tr>
<tr>
<td>4.0 ms Ethernet/IP, 2.0 ms Task</td>
<td>13.242</td>
<td>7.590</td>
<td>19.861</td>
<td>3.296</td>
<td>11%</td>
</tr>
<tr>
<td>2.0 ms Ethernet/IP, 6.0 ms Task</td>
<td>19.415</td>
<td>11.670</td>
<td>29.832</td>
<td>3.958</td>
<td>19%</td>
</tr>
<tr>
<td>4.0 ms Ethernet/IP, 6.0 ms Task</td>
<td>17.745</td>
<td>11.687</td>
<td>29.818</td>
<td>4.694</td>
<td>11%</td>
</tr>
</tbody>
</table>

These measurements were performed with the following programs scheduled:

• General

• 3_PosTable (myPosTable[0..9]) + some additional logic for the measurements

6.2 Memory Usage

The pictures below indicate the memory usage for:

• Only the Add-On Instructions (figure on the left side)

• Add-On Instructions and all example programs (figure on the right side)
HEAD OFFICE
2-1 Kurosakishiroishi, Yahatanishi-ku, Kitakyushu 806-0004, Japan
Phone  +81-93-645-7703  Fax  +81-93-645-7802

YASKAWA America Inc. (Motoman Robotics Division)
100 Automation Way, Miamisburg, OH 45342, U.S.A.
Phone  +1-937-847-6200  Fax  +1-937-847-6277

YASKAWA Europe GmbH (Robotics Division)
Yaskawastrasse 1, 85391 Allershausen, Germany
Phone  +49-8166-90-100  Fax  +49-8166-90-103

YASKAWA Nordic AB
Verkstadsgatan 2, Box 504, SE-385 25 Torsas, Sweden
Phone  +46-480-417-800  Fax  +46-486-414-10

YASKAWA Electric (China) Co., Ltd.
22F, One Corporate Avenue, No.222, Hubin Road, Huangpu District, Shanghai 200021, China
Phone  +86-21-5385-2200  Fax  +86-21-5385-3299

YASKAWA America Inc. (Motoman Robotics Division)
100 Automation Way, Miamisburg, OH 45342, U.S.A.
Phone  +1-937-847-6200  Fax  +1-937-847-6277

YASKAWA Electric (China) Co., Ltd.
22F, One Corporate Avenue, No.222, Hubin Road, Huangpu District, Shanghai 200021, China
Phone  +86-21-5385-2200  Fax  +86-21-5385-3299

YASKAWA Electric (Singapore) PTE Ltd.
151 Lorong Chuan, #04-02A, New Tech Park, Singapore 556741
Phone  +65-6282-3003  Fax  +65-6289-3003

YASKAWA Electric (Thailand) Co., Ltd.
59,1st-5th Floor, Flourish Building, Soi Ratchadapisek 18,Ratchadapisek Road,
Huaykwang, Bangkok 10310, THAILAND
Phone  +66-2-017-0099  Fax  +66-2-017-0199

Specifications are subject to change without notice
for ongoing product modifications and improvements.