
UNIVERSITY OF MESSINA

DEPARTMENT OF ENGINEERING

ENGINEERING AND COMPUTER SCIENCE

MASTER'S DEGREE

# Robotic arm

Fischertecnick robotic arm

Industrial automation and robotics mod.B

STUDENTS:
**Gianluca Catalfamo**
**Gabriele Morabito**
**Giorgio Nocera**
**Serena Sebbio**

TEACHER:
**Prof. Eng. Luca Patanè**

ACADEMIC YEAR 2020-2021

# Contents

CHAPTER 1

# Introduction

As part of the final exam of Industrial Automation and Robotics - part B, a Fischertechnick robotic arm has been controlled by a Siemens SIMATIC S7-1200 PLC through Siemens Totally Integrated Automation (TIA) Portal software environment.

## 1.1 Goal of the project

The goals of this project are:

- to manage to implement an high level control system of a robotic arm using MATLAB, TIA Portal and a Siemens SIMATIC S7-1200 PLC;
- to create a 3D model of the robot, to simulate its behaviour and to compute the direct and inverse kinematic;
- to validate the model performing a simple task in the real world: pick up an object, move it to another position and release it.

# CHAPTER 2

# Enabling technologies

## 2.1 Hardware

### 2.1.1 PLC SIMATIC S7-1200

The PLC SIMATIC S7-1200 with CPU 1215C AC/DC/RLY (Figure2.1) is a PLC produced by Siemens and it was used in this project in order to realize the sequence control of the robotic arm.
It is characterized by:

- 14 24V digital inputs;
- 10 relay outputs;
- 2 analog inputs;
- 2 analog outputs.

### 2.1.2 Fischertechnik 3D-Robot 24V

The 3D-Robot 24V (Figure2.2) is a robotic arm produced by Fischertechnik. It is a 3-DOF (3 degrees of freedom) robot and in particular it is composed by:

- a revolute joint;


**Figure 2.1:** Siemens SIMATIC S7-1200 1215C AC/DC/RLY PLC

- 2 prismatic joints (up/down and frontward/backward);
- a gripper as end-effector.

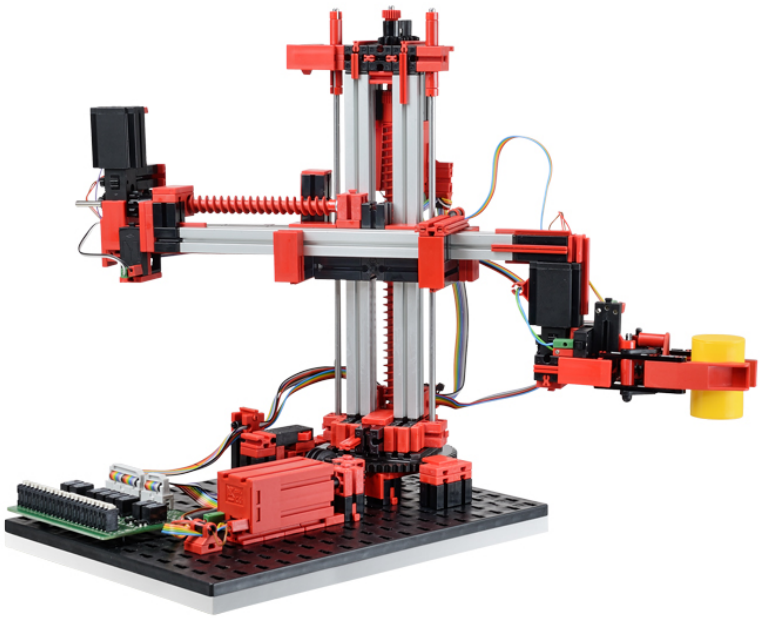


**Figure 2.2:** Fischertechnik 3D-Robot 24V

Thus the robot is characterized by a cylindrical geometry: its workspace is a portion of a hollow cylinder.

**Mini switches**

The robot has four mini switches (Figure2.3) that are pressed respectively in four different conditions:

- the gripper is fully open;
- the end-effector is completely brought back;
- the horizontal arm is completely brought up;
- the robot is completely rotated clockwise.

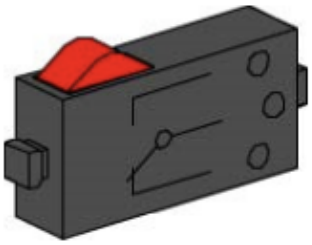


**Figure 2.3:** Mini switch

**Pulse counters**

The robot has two pulse counters, that can be used to understand respectively:

- how much the gripper is open;
- how much the horizontal arm is moved frontward.

In fact, the pulse counters are switches that are automatically pressed each time a gear involved in the corresponding movement rotate of a certain angle, producing a certain quantity of movement.

**Encoders**

The robot has two motor encoders (Figure2.4) with a maximum frequency of 1 KHz, that can be used to understand respectively:

- how much end horizontal arm is moved down;

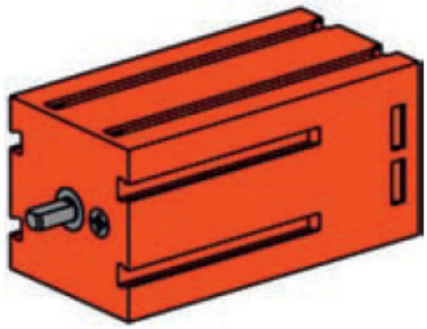


**Figure 2.4:** Motor encoder

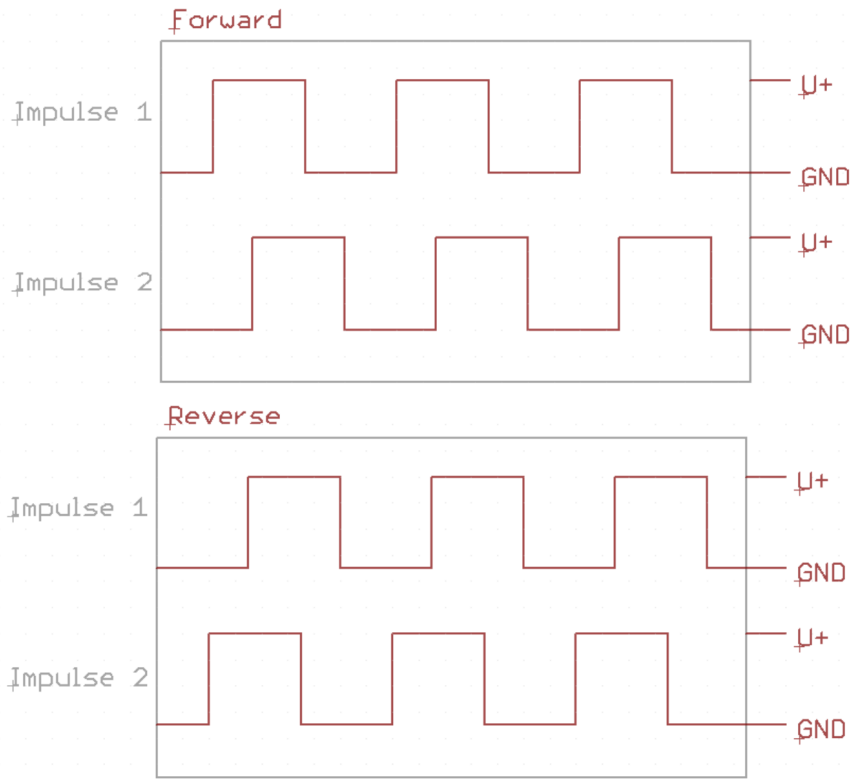



**Figure 2.5:** Encoder signals

- how much the robot is rotated.

The direction of the movement/rotation can be understood by the time order of the two signals generated by the encoder (Figure2.5).

### 2.1.3 Push-buttons

In this project, three LM2T metal push-buttons with spring return produced by Lovato Electronics are used:

- a black one (8 LM2T B102) with a normally open contact element is used for the reset operation that moves the robot to the initial position;
- a black one (8 LM2T B102) with a normally open contact element is used to start the TCP connection between the PLC and MATLAB;
- a red one (8 LM2T B104) with a normally closed contact element is used as a stop emergency button to stop the robot immediately.

**(a)** Push-button


**(b)** Normally open contact element


**(c)** Normally closed contact element

**Figure 2.6:** Push-buttons

### 2.1.4 Light Emitting Diodes

Two Light Emitting Diodes produced by Lovato Electronics are used in this project. Each LED is made of a LED integrated lamp-holder, a mounting adapter and a pilot light head. A green LED is turned on when the robot is ready to move and it blinks while the robot is moving. A red LED is turned on after the emergency stop button has been pressed and it blinks during the reset operation.

**(a)** Light integrated lamp-holder

**(b)** Mounting adapter

**(c)** Pilot light head

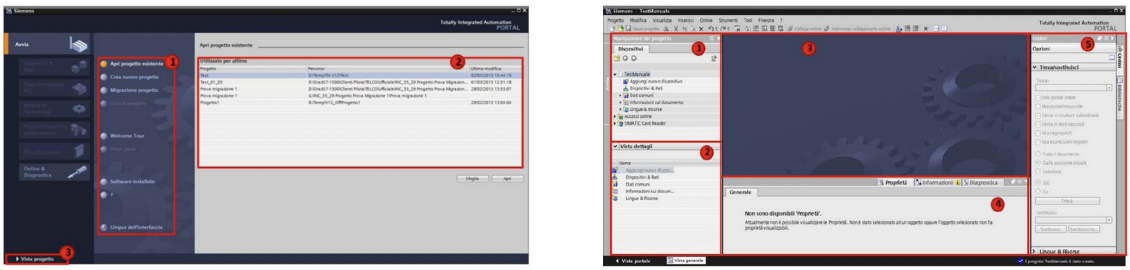
**Figure 2.7:** LED

## 2.2 Software

### 2.2.1 TIA Portal

TIA Portal (Totally Integrated Automation Portal) is a software package by Siemens created specifically to develop automation using Siemens products such as PLCs. It is in practice a centralized design environment characterized by a common user interface for all automation tasks with shared services (such as those of configuration, communication and diagnostics) and a single database to which also other software packages, such as SIMATIC WinCC V12, SINAMICS Startdrive V12 and SIMATIC STEP 7 PLCSIM V12, access. The version 15 of the software was used in this project in order to design and upload the control program in the PLC. TIA Portal has a user interface characterized by the presence of two views:

- the portal view;
- the project view.



**(a)** Portal view **(b)** Project view

**Figure 2.8:** TIA Portal

#### Portal view

The portal view is the one that opens automatically when it is launched the TIA Portal and that allows the user to choose which operations he wants to perform with the TIA Portal. It is characterized by the presence of:

1. a window where it is possible to choose which operation you want to perform;
2. a selection window related to the selected operation;
3. a button that allows to switch to the project view.

**Project view**

The project view is the working window of the TIA portal that allows the performance of any function within a project; from the project view it is possible to access all the components of the project and quickly navigate within it. It is characterized by the presence of:

1. a window where it is possible to access and navigate all the components of the project;
2. a window where the content of the component selected in window 1 is visualized;
3. a window that allows the user to make changes to the project: the editors for writing of the software, the definition of the hardware or the definition of the panel pages based on the context in which the user is located are displayed;
4. a window where it is possible to view the properties and the details of the objects selected in the window 3;
5. a window that varies according to the editor that comes presented in window 3 and allows to view and use the TIA Portal Libraries tool.

### 2.2.2 MATLAB

MATLAB (an abbreviation of "matrix laboratory") is a programming and numeric computing platform developed by MathWorks. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, data analysis, creation of user interfaces, and interfacing with programs written in other languages. After starting MATLAB, the desktop appears in its default layout.

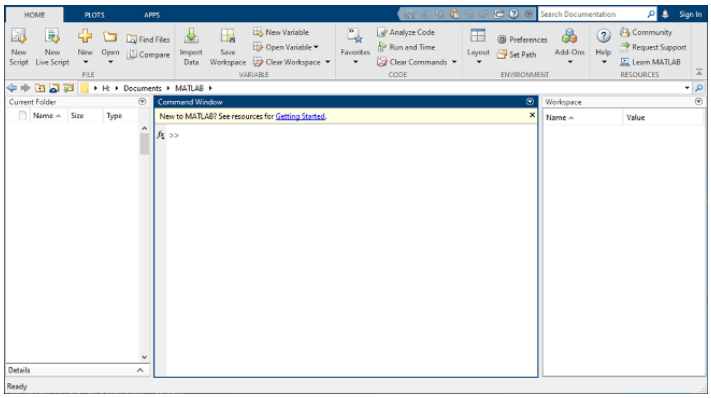



**Figure 2.9:** Matlab desktop

The desktop includes these panels:

- Current Folder — for file access
- Command Window — for entering commands at the command line, indicated by the prompt (»)
- Workspace — for exploring data created or imported from files. [3]

In this project a TCP communication between MATLAB (R2021a) and TIA Portal is established to send and receive data continuously.

### 2.2.3 CoppeliaSim

CoppeliaSim, formerly known as V-REP (Virtual Robot Experimentation Platform), is a general purpose robot simulator, with integrated development environment, used in industry, education and research for fast algorithm development, factory automation simulations, fast prototyping and verification, robotics related education, motion planning, remote monitoring and safety double-checking. CoppeliaSim is based on a distributed control architecture: each object/model can be individually controlled via an embedded script, a plugin, a ROS or BlueZero node, a remote API client, or a custom solution. This makes CoppeliaSim very versatile and ideal for multi-robot applications. Controllers can be written in C/C++ (plug-ins), Lua (scripts acting as individual synchronous controllers), Python, Java, Matlab or Octave (asynchronous controllers). [5] At its core, CoppeliaSim uses a kinematics engine for forward and inverse kinematics calculations, and several physics simulation libraries to perform rigid body simulation. Models and scenes are built by assembling various objects (meshes, joints, various sensors, Point clouds, OC trees, etc.) into a hierarchial structure. Additional functionality, provided by plug-ins, include: motion planning (via OMPL), synthetic vision and imaging processing (e.g. via OpenCV), collision detection, minimum distance calculation, custom graphical user interfaces and Data visualization (e.g. via plots). [7]

Scene objects are basic building blocks that can be combined with each other. They can form complex systems together with calculation modules and control mechanisms.

There are fourteen types of scene objects: proximity sensors, graphs, vision sensors, paths, mills, cameras, lights, mirrors, shapes, joints, force / torque sensors, dummies, octrees and point clouds.

There are five calculation modules:

- Collision detection
- Physics / Dynamics
- Minimum distance calculation
- Path / Motion planning
- Forward / Inverse kinematics

# CHAPTER 3

# Implementation

## 3.1 Analisys

To analyze the behaviour of the robot, a first step is related to its kinematic. Kinematic analysis of the mechanical structure of a robot concerns the description of the motion with respect to a fixed reference Cartesian frame by ignoring the forces and moments that cause motion of the structure. Independently by the weight, the kinematic motion is the same. The weight, instead, is important when one want to evaluate which is the force needed to perform an action, the power to provide to the motor, and this is covered by the dynamical analysis. The kinematic describes the analytical relationship between the joint positions and the end-effector position and orientation.

Since the goal of the project is to manipulate an object in space, it is necessary to describe the end-effector position and orientation (pose) as a function of the joint variables of the mechanical structure with respect to a reference frame.

### 3.1.1 Direct Kinematics

The mechanical structure of a manipulator is characterized by a number of degrees of freedom (DOFs) which uniquely determine its posture, i.e. the pose of all the rigid bodies composing the chain. Each DOF is typically associated with a joint articulation and constitutes

a joint variable.

In particular, the robot used in this project is a 3 DOFs manipulator composed by a revolute joint and two prismatic joints. A 2D representation of its structure is reported in Figure3.1.

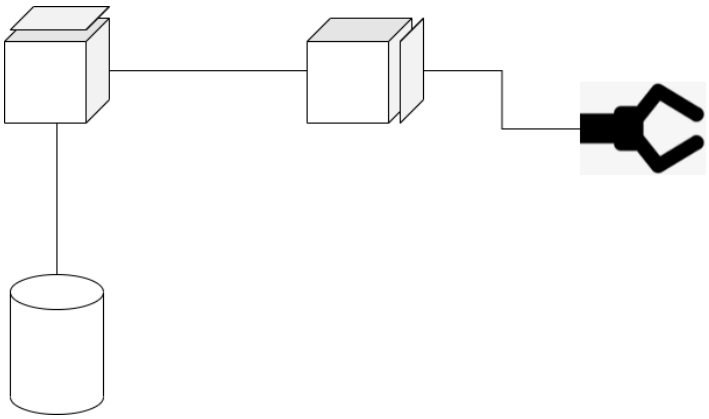


**Figure 3.1:** 2D schematic representation of the robot

To apply a kinematic analysis let's consider an open-chain manipulator constituted by $n+1$ links connected by $n$ joints, where Link 0 is conventionally fixed to the ground and assume that each joint provides the mechanical structure with a single DOF, corresponding to the joint variable. Defining a suitable coordinate frame attached to each link, from Link 0 to Link n the coordinate transformation describing the position and orientation of Frame n with respect to Frame 0 is given by:

$$T_n^0(q) = A_1^0(q1)A_2^1(q2)...A_n^{n-1}(qn)$$

A first way to compute direct kinematics is offered by a geometric analysis of the structure of the given manipulator. The effectiveness of a geometric approach to the direct kinematics problem is based on the ability of the person that creates and analyze the model and, in the choice of the correct and most suitable quantities to define the robot properties. Anyway, if the manipulator structure is complex and the number of joints increases, it is preferable to adopt a less direct solution based on a systematic, general procedure known as Denavit-Hartenberg Method.

### 3.1.2 Denavit-Hartenberg Method

In order to compute the direct kinematics equation for an open-chain manipulator the Denavit-Hartenberg method has been derived. It is based on four steps that has to be performed to obtain the homogeneous transformation matrix, through which it is possible to know the position and orientation of the end effector.

- Assign frames according to the four DH rules.
- Create the DH parameter table.
- Plug the table values into the Homogeneous Transformation Matrix.
- Multiply the matrices together applying the chaining rule.

#### Frames Assignment

The goal is to define the relative position and orientation of each pair of consecutive links. The problem is that of determining two frames attached to the two links and computing the coordinate transformations between them. The easiest solution is to define some rules to be followed to define in a specific way the frames, so that the final mathematical representation can be as simple as possible. These rules are based on the Denavit-Hartenberg convention. To define the frame for a generic link i (frames are always attached to the link and not to the joint), the following passages has to be realized:

- Choose axis $z_i$ along the axis of Joint $i + 1$.
- Locate the origin $O_i$ at the intersection of axis $z_i$ with the common normal to axes $z_i - 1$ and $z_i$. Also, locate $O_i'$ at the intersection of the common normal with axis $z_i - 1$.
- Choose axis $x_i$ along the common normal to axes $z_i - 1$ and $z_i$ with positive direction from Joint i to Joint i+ 1.
- Choose axis $y_i$ so as to complete a right-handed frame.

When defining the frame, notice that it is not true that the reference frame attached to the link has to be in the middle of the object. It could be inside the object but it could be also outside from the physical shape of the object. The only important thing is that the movement

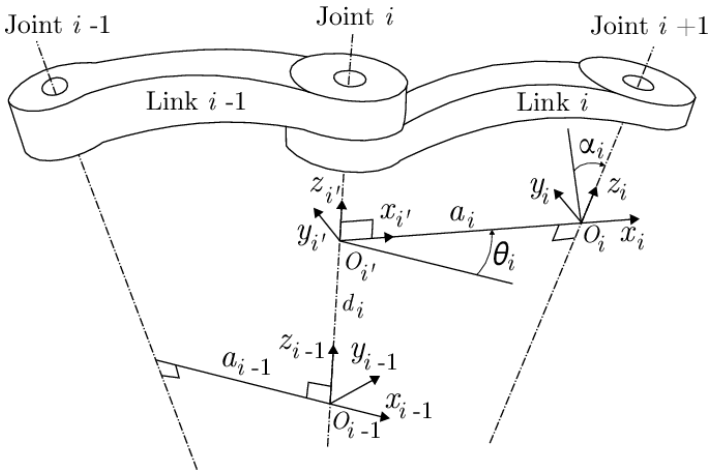



**Figure 3.2:** Denavit-Hartenberg convention for frame assignment

of the links is integral with the reference frame, that means that the movements have an effect on the reference frame, i.e. they move in the same way.

The four rules can be rewritten also in a simpler way as follows:

- The $Z$ axis must be the axis of revolution, for revolute joint, or the direction of motion, for prismatic joint.
- The $X$ axis must be perpendicular to the $Z$ axis of the frame before it.
- The $X$ axis must intersect the $Z$ axis of the frame before it.
- The $Y$ axis must be drawn so the whole frame follows the right-hand rule.

It is also important to notice that are needed at least one more frame than there are joints, since one frame must be on the end-effector. Eventually, it is possible to have also more frames but increasing the number of frames increase also the complexity of the mathematical representation. Before to assign the frame to the link, consider that all axes must be drawn either up, down, right, left, or in the first or third quadrant. This convention is used to obtain a coherent 2D representation in which the axis in the first quadrant represents an axis that goes into the page, whereas the axis in the third quadrant represents an axis that goes out of the page.

Each reference frame is attached to the previous link, so the first one refers to the base of the robot and can be considered as a general fixed reference frame. In order to follow the first Denavit-Hartenberg frame rule the $Z_0$ axis has been assigned so that it coincides with the

rotation axis of the revolute joint. For the $X_0$ axis since, in this case, there is not a previous frame it is possible to assign it arbitrarily, without any constraint. Finally, the $Y_0$ axis has to be drawn following the right hand rule, according to the fourth DH frame rule. The final result for the first frame is shown in Figure3.3.

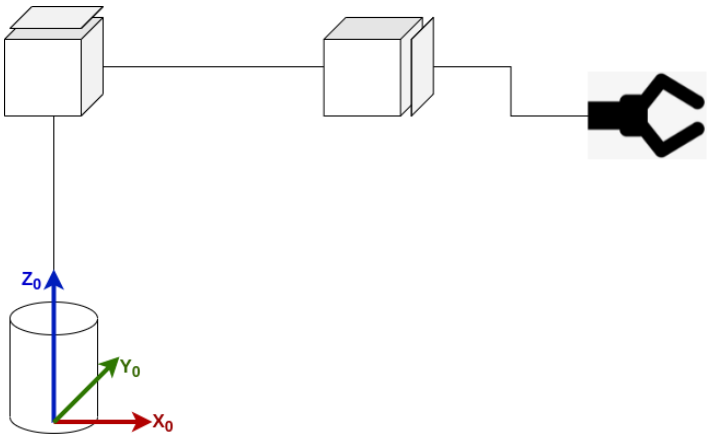



**Figure 3.3:** First reference frame

For the second frame, the $Z_1$ axis follows the direction of motion of the prismatic joint and it points up. To define the direction of the $X_1$ now the second and third rule has to be satisfied. For this reason, it has been chosen toward right so that it is perpendicular to $Z_0$ and, prolonging $Z_0$ and $X_1$, they intersect. The $Y_1$ axis follows the right hand rule. The final result for the second frame is shown in Figure3.4.

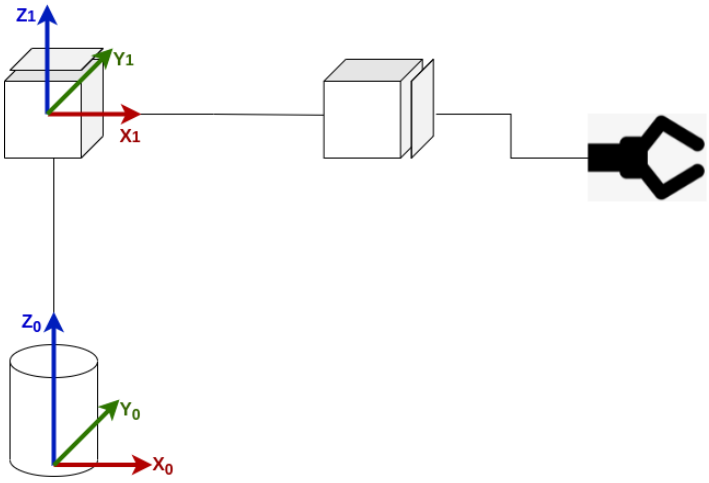



**Figure 3.4:** Second reference frame

In the third frame, the $Z_2$ axis follows the direction of motion of the second prismatic joint. Then, since the $X_2$ axis must be perpendicular to the $Z_1$ axis, it can not be drawn up or

down because in this case it would be parallel and can not be drawn right or left because it has to be perpendicular also to its own $Z_2$ axis. So, it points out of the page. However, it does not respect the third DH rule because the $X_2$ axis and the $Z_1$ axis does not intersect. To solve this kind of problem, it is possible to translate the reference frame, as a first option, and if this does not solve the issue a rotation of the frame has to be applied. The first solution is preferable because it does not change the mutual position of the current frame with the previous ones. In this case translating the third reference frame, so that its origin coincides with the one of the second frame, is enough to solve the problem. The $Y_2$ axis follows the right hand rule. The final result for the third frame is shown in Figure3.5.

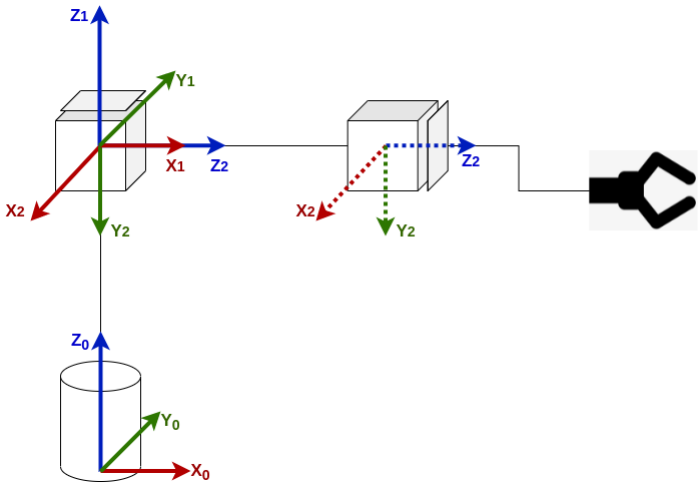



**Figure 3.5:** Third reference frame

The last frame is attached to the end effector, and is generally addressed as end-effector frame. Here, there is not a constraint for the $Z_3$ so the best solution is to define it parallel to the previous frame to keep simple the maths. The $X_3$ axis, instead, must be perpendicular to $Z_2$ and should intersect it, so it has been drawn up. The $Y_3$ axis follows the right hand rule. The complete model with all the frames correctly assigned is shown in Figure3.6.

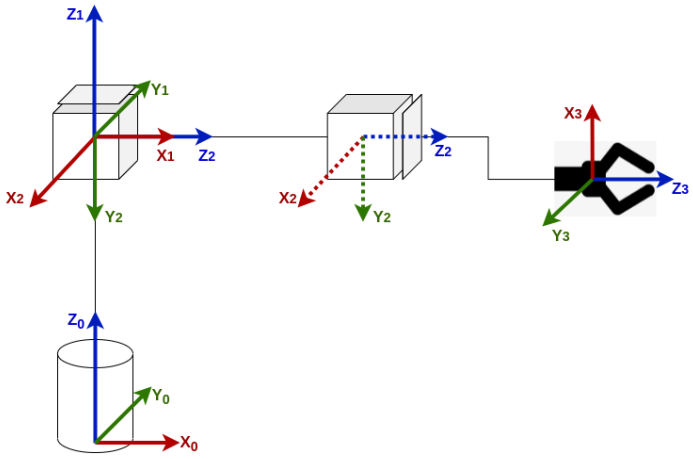



**Figure 3.6:** Assigned reference frames

### DH parameter table

The second step of the Denavit-Hartenberg method regards the evaluation of the parameters of the robot and the creation of the corresponding DH parameter table. The DH parameter table records all of the displacement and rotation relationship between each pair of frames. As already said, this is important to identify the kinematic of a robot that can be obtained chaining the relation between the reference frames, that are related to translations and rotations. Inside the table there are a certain number of columns and rows. The columns are always four: two are related to rotations and the other two are related to displacements. The number of rows, instead, varies according to the analyzed robot. In particular, if N is the number of frames, the number of rows will be N-1 just because each row will represent information that put in relation two adjacent frames. In this case, there are four frames so the table will have three rows. The application of the DH convention described before allows to minimize the number of parameters needed. Actually, only two parameters for the rotation and two parameters for the displacement are needed. So, having this parameters for each pair of frames the homogeneous transformation matrix can be directly derived.

In particular, the four parameters are:

- $\theta$. It is the amount we have to rotate frame n-1 around the axis $Z_n - 1$ in order to get axis $X_n - 1$ to match the axis $X_n$.

  Moreover, any rotation of the joint has to be included if the joint is a revolute one, because this rotation is also a rotation around the axis $Z_n - 1$. The value of $\theta$ is composed

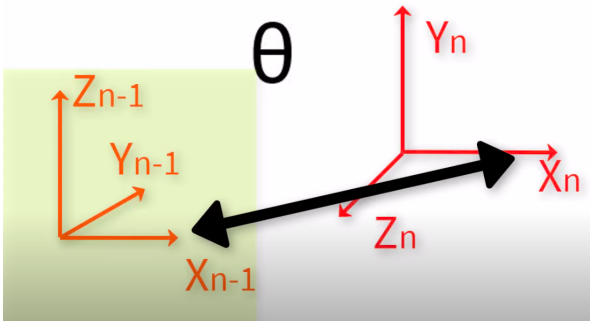



**Figure 3.7:** First rotational parameter

by two elements:

- A static part. For construction we could have a static offset, so in this case a rotation to align the two X axis has to be applied.
- A dynamic part. If the frame is placed on a revolute joint also the rotation has to be considered. If the revolute joint rotates, it will change the orientation of each link, and the orientation of each reference frame attached to the other elements of the robot, and this could create a relative angle between the two adjacent axis.

At the end, the correct value is obtained adding the static offset and the dynamic effect of the joint. This last contribute is coincident with the angle of rotation of the joint that is considered to be positive if the rotation is counter-clockwise.

- $\alpha$. The angle of rotation of frame n-1 around $X_n$ in order to align the two $Z$ axis.

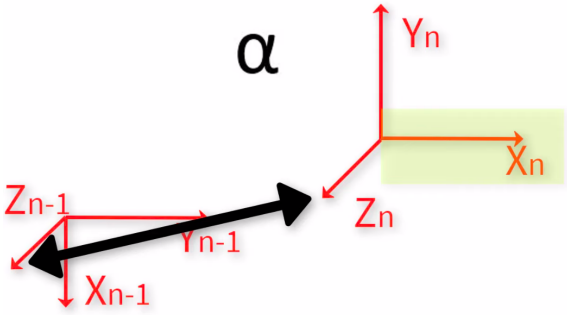



**Figure 3.8:** Second rotational parameter

The axis $X_n$ is fixed and the plane $X_{n-1}Z_{n-1}$ rotates in order to align $Z_n$ and $Z_{n-1}$. To know the sign of the rotation angle the right hand rule is used. So, putting the thumb in the direction of $X$ (now the $X$ is the rotation axis) it is possible to verify in

which direction the other fingers, that are aligned with the $Z_{n-1}$ rotate. If the rotation is performed clockwise the sign is negative, if counterclockwise the value of $\alpha$ is positive. For the alpha column, the rotation due to the actuator does not produce any effect on the mutual orientation between the frames, so it is not needed to consider the dynamic part related to the effect of the rotation of the revolute joint.

- **r**. The first displacement parameter is the amount of displacement from the centre of the reference frame n-1 to the centre of the frame n, measured only in the $X_n$ direction.

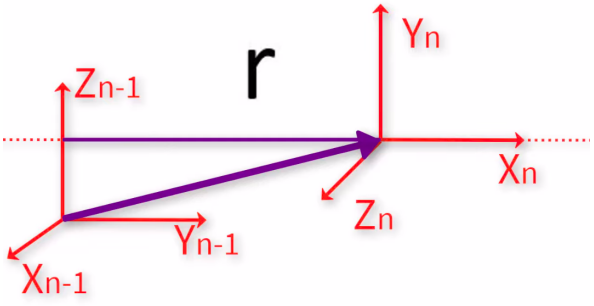



**Figure 3.9:** First displacement parameter

- **d**. It is the amount of displacement from the centre of frame n-1 to the centre of frame n, measured only in the $Z_n - 1$ direction.

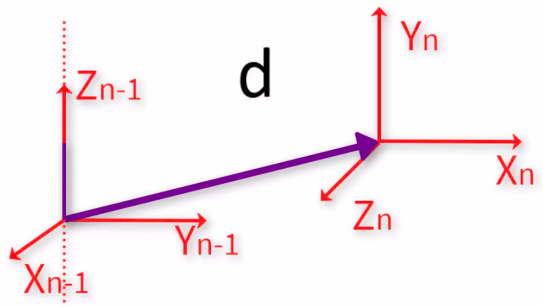



**Figure 3.10:** Second displacement parameter

For our robot, the DH parameter table is the following:

| $\theta$ | $\alpha$ | **r** | **d** |
|---|---|---|---|
| $0 + \theta_1$ | 0 | 0 | $a_1$ |
| $-90°$ | $-90°$ | 0 | $0 + d_1$ |
| $-90°$ | 0 | $a_4$ | $a_2 + a_3 + d_2$ |

**Table 3.1:** Denavit-Hartenberg parameter table

Let's consider the parameter $\theta$. The rotation of frame zero around $Z_0$ to align $X_0$ and $X_1$ is equal to zero since they are already aligned, but the dynamic component related to the rotation of the revolute joint ($\theta_1$) is then added. The rotation of frame 1 around $Z_1$ in order to get axis $X_1$ to match the axis $X_2$ is equal to $90°$. The sign is negative because we rotate clockwise. Also the rotation of frame 2 around $Z_2$ in order to get axis $X_2$ to match the axis $X_3$ is equal to $-90°$.

The $\alpha$ parameters have been derived as follows. The angle of rotation of frame zero around $X_1$ in order to align the two $Z$ axes is zero because they are already aligned. The angle of rotation of frame 1 around $X_2$ to align $Z_1$ and $Z_2$ is $-90°$ because the rotation is clockwise. Then, axes $Z_2$ and $Z_3$ are aligned, so the last angle $\alpha$ is zero.

Considering the $r$ parameter, the displacement between the origin of frame 0 and frame 1 measured along the $X_n$ is zero. The distance between the origins of frame 1 and frame 2 is zero because they coincide. The displacement between the origin of frame 2 and the origin of frame 3 is reported in figure 3.11 as an orange arrow. Its projection on the $X_3$ axis is the segment indicated with $a_4$.

The displacement $d$ between frame 0 and frame 1 measured on the $Z_0$ axis is the segment $a_1$ (see figure 3.11). The distance between $O_1$ and $O_2$ is null because they coincide but the dynamic component $d_1$ related to the prismatic joint motion has to be added. The distance between $O_2$ and $O_3$ along $Z_2$ is the sum of $a_2$ (dimension of the second link), $a_3$ (dimension of the third link) and $d_2$ that is the dynamic component of the second prismatic joint.

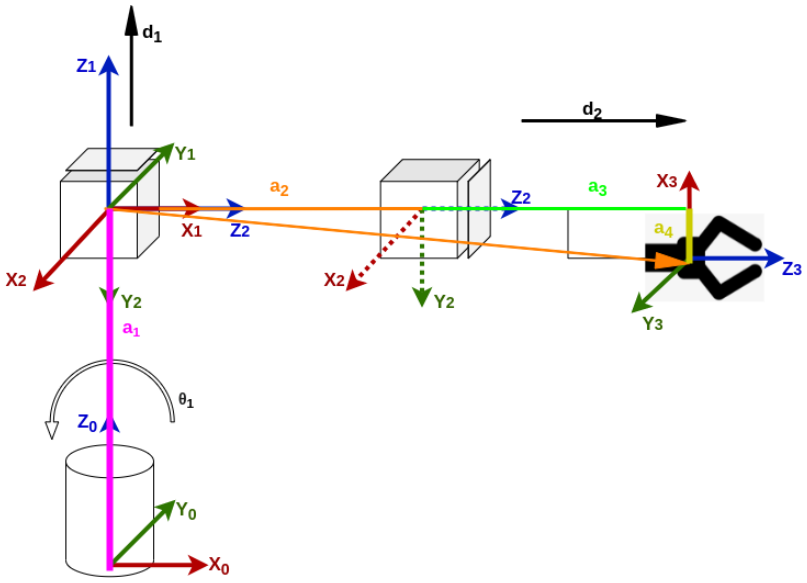



**Figure 3.11:** DH parameter

Substituting the value physically measured on the robot we obtain the table 3.2.

| $\theta$ | $\alpha$ | **r** (cm) | **d** (cm) |
|---|---|---|---|
| $\theta_1$ | 0 | 0 | 18.5 |
| $-90°$ | $-90°$ | 0 | $d_1$ |
| $-90°$ | 0 | 6.75 | $20 + d_2$ |

**Table 3.2:** Denavit-Hartenberg parameter table

The length of the second link $a_2$ is equal to zero since the third frame has been moved in the same point of the second frame. The link between the second prismatic joint and the gripper is around 20 cm.

### Homogeneous Transformation Matrix

In order to achieve a compact representation of the relationship between the coordinates of the same point in two different frames, the homogeneous representation of a generic vector p can be introduced as a new vector formed by adding a fourth unit component. This new way to represent the points allows to avoid in the mapping between two generic points p0 and p1, the presence of an additive term and it is possible just to condensate this mapping into a new matrix A, the homogeneous transformation matrix.

The shape of the new matrix is a block matrix that contains a first block that is the rotation matrix, on the right side there is an other block that is a column that contains the three components of the vector $O_1^0$ that is the displacement between the two origins. The last row is composed by a vector of zeros and a term equal to one.

At this point, it is possible to express the coordinate transformation between each pair of frame $i$ and $i-1$.

$$A_i^{i-1}(q_i) = A_i'^{i-1} A_i^{i'} = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i} c_{\alpha_i} & s_{\theta_i} s_{\alpha_i} & r_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i} c_{\alpha_i} & -c_{\theta_i} s_{\alpha_i} & r_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1.1}$$

Matrix frame 0 frame 1.

$$A_1^0(q_1) = \begin{bmatrix} c_{\theta_1} & -s_{\theta_1} & 0 & 0 \\ s_{\theta_1} & c_{\theta_1} & 0 & 0 \\ 0 & 0 & 1 & 18.5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1.2}$$

Matrix frame 1 frame 2.

$$A_2^1(q_2) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1.3}$$

Matrix frame 2 frame 3.

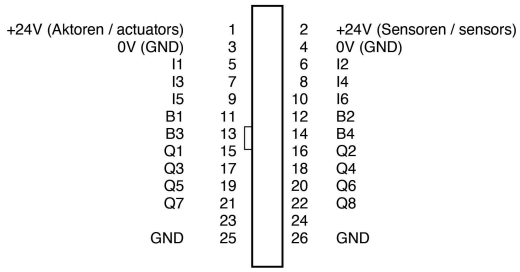
$$A_3^2(q_3) = \begin{bmatrix} r0 & 1 & 0 & 0 \\ -1 & 0 & 0 & -6.75 \\ 0 & -1 & 0 & 20 + d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1.4}$$

Complete matrix from frame 0 to frame 3

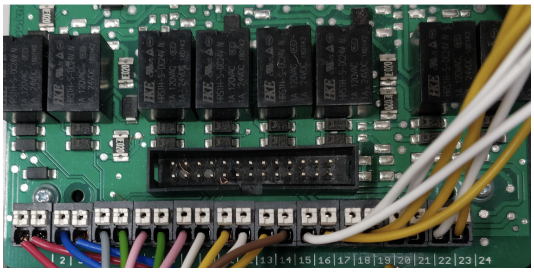
$$A_3^0(q_3) = \begin{bmatrix} 0 & s_{\theta_1} - c_{\theta_1} & 0 & c_{\theta_1} d_1 \\ 0 & -c_{\theta_1} - s_{\theta_1} & 0 & d_1 s_{\theta_1} \\ 1 & 0 & 0 & d_1 + 13.5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1.5}$$

## 3.2 Hardware setup

The PLC is used to control the Robot. In particular, the robot use a pinout of 24 pins for inputs and outputs. The pinout, which is shown in Figure3.12a, describes all the pins which are connected to a particular element of the robot.




**(a)** Robot Pinout



**(b)** Photo of Robot Pinout

**Figure 3.12:** TIA Portal

A more detailed description is presented in the table 3.3, which explains exactly which is the role of each pin. They are detailed the position, a description, the name and the type of pin (input or output). Moreover, from this table it is also possible to understand how to control each movement of the robot and how to get feedback from it. For example, by powering the terminal number **15**, it is possible to open the gripper (the end effector) of the robot, while by looking at the status of the pin **5**, it is possible to know if the gripper is fully open or not.

In order to control the robot, the PLC needs to be connected to it. As it is intuitive, the output of the robot (which are useful as feedback for the operations) are connected to the PLC inputs and viceversa. The connection between the two electronic components is depicted with more details in the schematic in Figure3.13 realized with KiCad, a free software suite for electronic design automation (EDA).

**Table 3.3:** Robot pinout description

| Terminal No. | Function | Pin Name | Robot Input/Output |
|---|---|---|---|
| 1 | Power Supply (+) | 24V DC | OUTPUT |
| 2 | Power Supply (+) | 24V DC | OUTPUT |
| 3 | Power Supply (-) | 0 | OUTPUT |
| 4 | Power Supply (-) | 0 | OUTPUT |
| 5 | Fully Open Gripper | I1 | OUTPUT |
| 6 | Pulse Counter Gripper | I2 | OUTPUT |
| 7 | Fully Back Arm | I3 | OUTPUT |
| 8 | Pulse Counter Back Arm | I4 | OUTPUT |
| 9 | Fully Top Arm | I5 | OUTPUT |
| 10 | Fully Clockwise Arm | I6 | OUTPUT |
| 11 | Encoder Vertical Axis impulse 1 | B1 | OUTPUT |
| 12 | Encoder Vertical Axis impulse 2 | B2 | OUTPUT |
| 13 | Encoder Turnable Axis impulse 1 | B3 | OUTPUT |
| 14 | Encoder Turnable Axis impulse 2 | B4 | OUTPUT |
| 15 | Motor Gripper Open | Q1 | INPUT |
| 16 | Motor Gripper Close | Q2 | INPUT |
| 17 | Motor Arm Front | Q3 | INPUT |
| 18 | Motor Arm Back | Q4 | INPUT |
| 19 | Motor Arm Down | Q5 | INPUT |
| 20 | Motor Arm Up | Q6 | INPUT |
| 21 | Motor Rotation Clockwise | Q7 | INPUT |
| 22 | Motor Rotation Counter Clockwise | Q8 | INPUT |

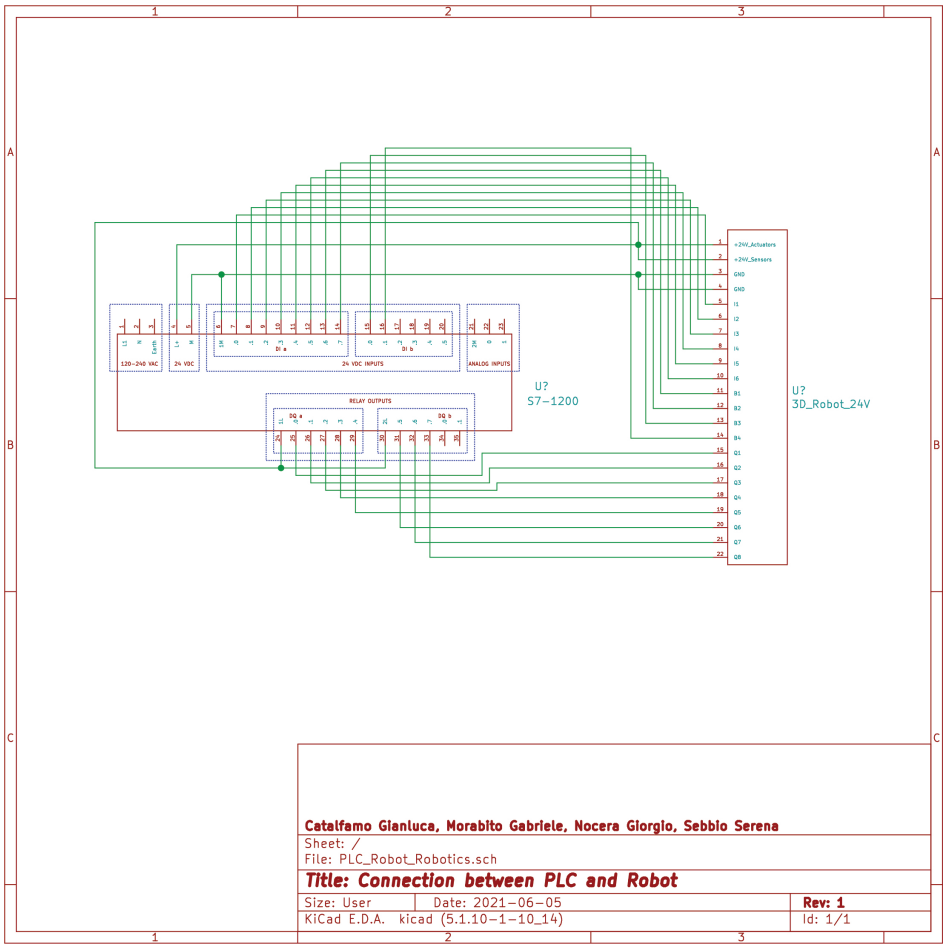



**Figure 3.13:** Connection between PLC and Robot

### 3.2.1 First prototype

For the first version of the prototype the connection shown in Figure3.13 was integrated with a set of 4 buttons, directly connected to the inputs pins *DIb .2* to *.5* of the PLC. This 4 buttons allowed the generation of 16 ($2^4$) possible combinations and, consequently, the possibility to send 16 different signals to the PLC by using only 4 inputs. Even though this system makes them available sixteen combinations, only eight of them were needed to address all the possible moves (Q1 to Q8 on the table 3.3). All the combinations are indicated in the table 3.4, where under the buttons columns, a value equal to 1 stands for button pressed and viceversa with a value of 0. This setup can only be considered as good as a development setup, and it allows to test sensors, motors and combinations before starting the design of the robotic actions.

**Table 3.4:** Moves combinations

| ID | Button 1 | Button 2 | Button 3 | Button 4 | Operation |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | None |
| 2 | 0 | 0 | 0 | 1 | Gripper Open |
| 3 | 0 | 0 | 1 | 0 | Gripper Close |
| 4 | 0 | 0 | 1 | 1 | Arm Front |
| 5 | 0 | 1 | 0 | 0 | Arm Back |
| 6 | 0 | 1 | 0 | 1 | Arm down |
| 7 | 0 | 1 | 1 | 0 | Arm Up |
| 8 | 0 | 1 | 1 | 1 | Rotation Clockwise |
| 9 | 1 | 0 | 0 | 0 | Rotation Counter Clockwise |
| 10 | 1 | 0 | 0 | 1 | None |
| 11 | 1 | 0 | 1 | 0 | None |
| 12 | 1 | 0 | 1 | 1 | None |
| 13 | 1 | 1 | 0 | 0 | None |
| 14 | 1 | 1 | 0 | 1 | None |
| 15 | 1 | 1 | 1 | 0 | None |
| 16 | 1 | 1 | 1 | 1 | Reset |

In Figure 3.14 it is possible to see the first prototype schematic.

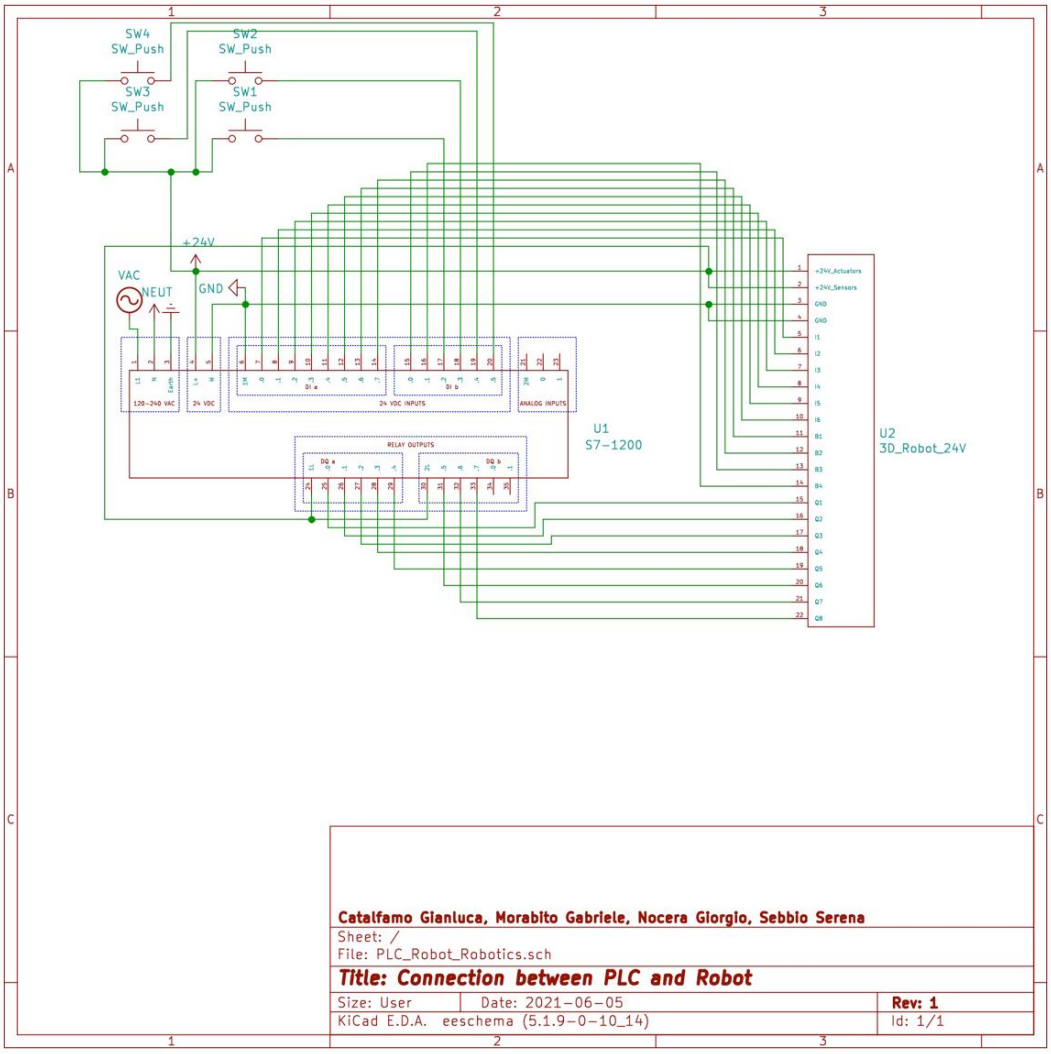
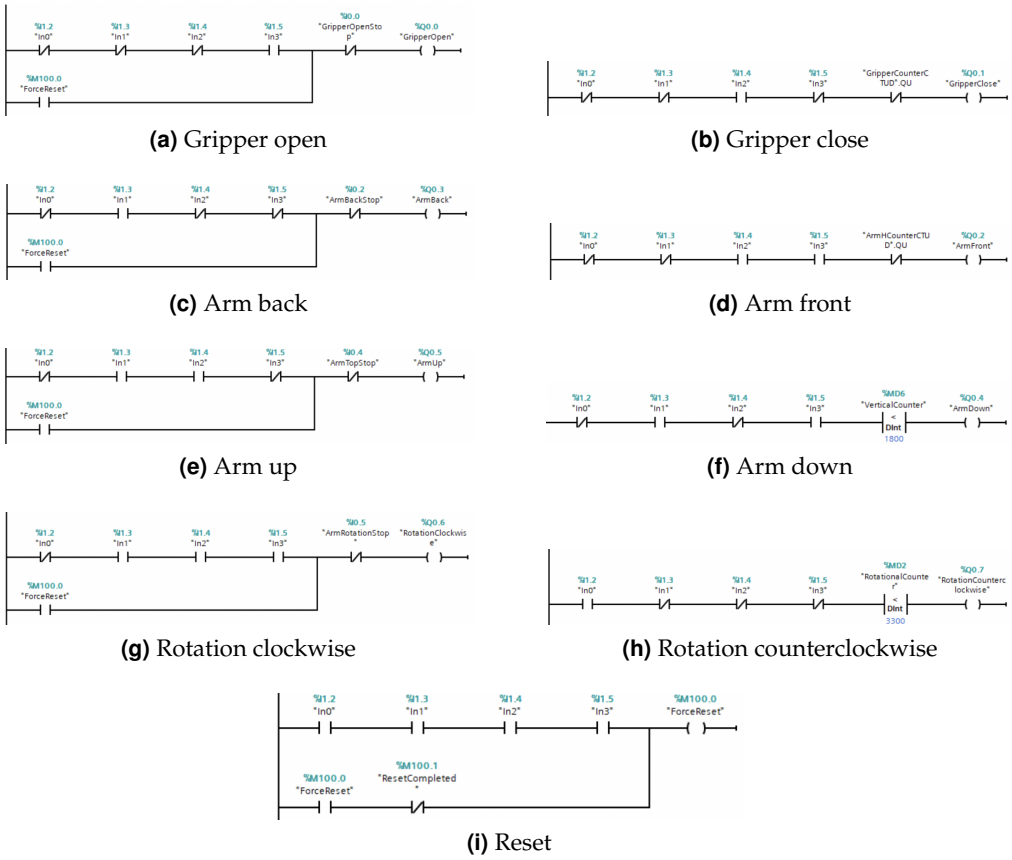



**Figure 3.14:** First prototype schematic

From Figure3.15 it is possible to see the working conditions explained before. In particular:

- In0, In1, In2, In3 correspond respectively to button 1, button 2, button 3 and button 4;
- GripperOpenStop, ArmBackStop, ArmTopStop and ArmRotationStop are the variables associated to the corresponding limit switch activation;
- the forward movement and the closure of the gripper are stopped if the motor is forced three times, while the counterclockwise rotation and the vertical down movement are stopped when the workspace limit is reached. The details about these stop conditions are explained in section 3.4.2, because they were used also in the final implementation.




**(a)** Gripper open

**(b)** Gripper close

**(c)** Arm back

**(d)** Arm front

**(e)** Arm up

**(f)** Arm down

**(g)** Rotation clockwise

**(h)** Rotation counterclockwise

**(i)** Reset

**Figure 3.15:** Prototype Ladder Diagram

**N.B.:** This implementation was realized in order to explore all the functionalities of the robotic arm and of the sensors embedded in it, before realizing the final hardware configuration and programming the PLC.

### 3.2.2 Final Schematic

The PLC, the robotic arm, the LEDs and the buttons were connected together following the schema in Figure 3.16

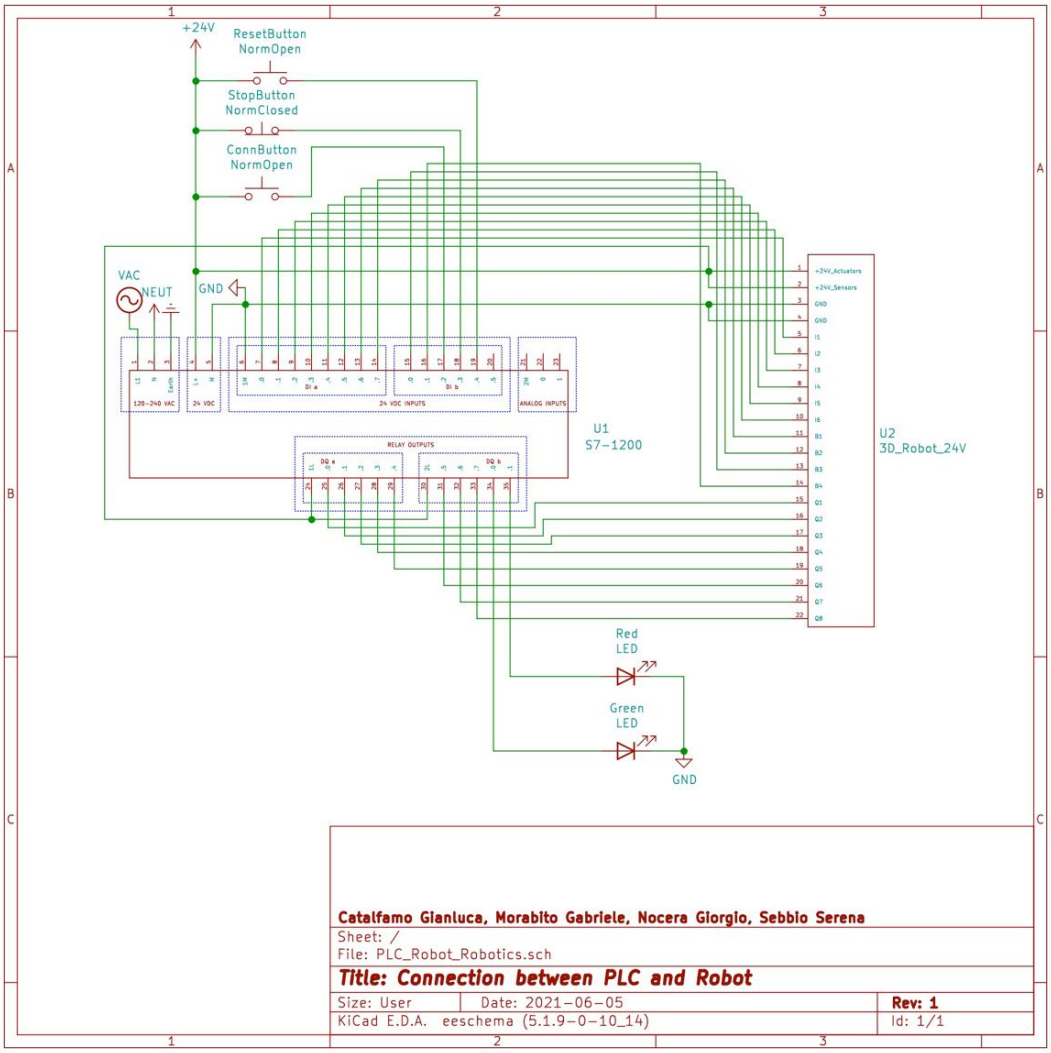



**Figure 3.16:** Final schematic

## 3.3 Environment setup

Once the hardware configuration has been completed, to control the movements of the robotic arm a connection between MATLAB and Tia Portal has been set up. This allow a bi-directional communication between the two software so that, the instructions can be launched directly from the command window using some predefined functions and then received and executed by the PLC that send the correct signals to the robotic arm.

### 3.3.1 Establish a TCP connection

The TCP connection established requires a master and a slave. In this case MATLAB is master and the PLC is the slave.

#### Tia Portal

First, the correct blocks need to be added in the ladder diagram to control the connection, the disconnection, the send and receive operations. In order to accomplish this task, with the version 15 of Tia Portal, a new segment is added for each of the following blocks:

- TCON (Figure3.17). The connection block is needed to establish the communication. To add this, go to the "Instructions" tab on the right sidebar, from the list choose the "Communication" section and then from the subsection "Others" drag and drop the TCON block in the newly created segment. Then, right click on the block and choose "Properties" to set the parameters. In the "Configuration" card, for the partner, choose the unspecified option; in the "Connection data" drop down, select new and let the default value in all the other fields. So, the connection ID used in this project, that has to be equal for all the subsequent blocks referring to the same connection will be the default value 1. The port will be the number 2000. Then specify the IP address of the PLC partner, i.e. the address of the computer running the MATLAB instance. To do this, physically connect the PLC to the computer with the Ethernet cable and start the connection from Tia Portal software. Then, the "ipconfig" command has been launched in the command prompt to retrieve the ip address. For this project, the ip address of the PLC is 192.168.0.1 and the one for the computer is 192.168.0.241. To the input port REQ of the TCON block a memory address has to be assigned so, define a bool variable inside the variable table choosing the first address one not already used and then map the tag to the input port.

- TDISCON (3.18). This function is used for terminating a TCP connection. Drag and drop the block from the same section seen for the connection function and define in the variable table the tag fro the REQ input that will be again a bool variable with a different and subsequent memory address with respect to the one defined for the connection function. Keep the same value for the connection ID.

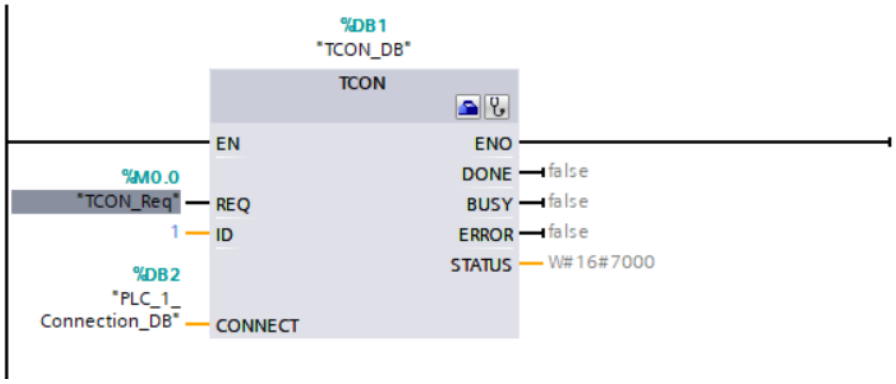



**Figure 3.17:** TCON function for the TCP connection

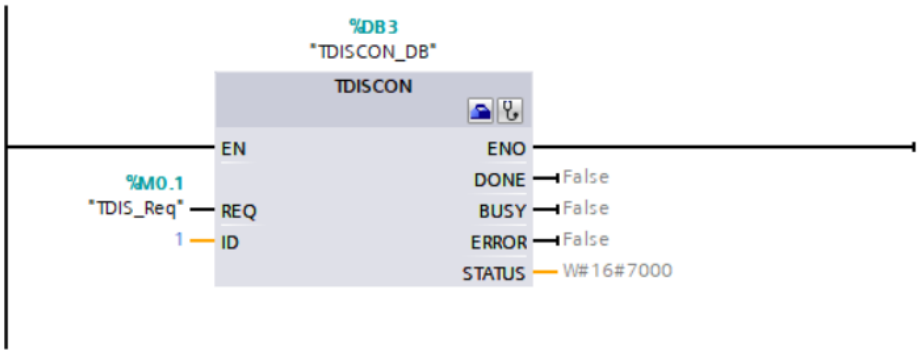



**Figure 3.18:** TDISCON function the for TCP connection

- TSEND (Figure3.19). The function that sends the data from PLC to the TCP partner. For the REQ input port let's define the bool variable with its own memory address. The input port "DATA" is the source from which data is sent.

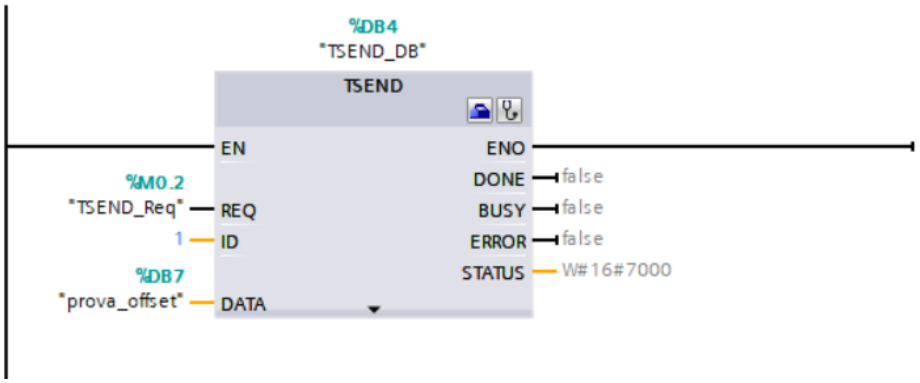



**Figure 3.19:** TSEND function the for TCP connection

- TRCV (Figure3.20). The function through which it is possible to receive data from partner, that is our MATLAB running program. Drag and drop the block, set the correct

connection ID, define a bool variable with a memory address and assign the tag to the REQ input port. The input port "DATA" of this function is the place where the received data from the PLC is stored.

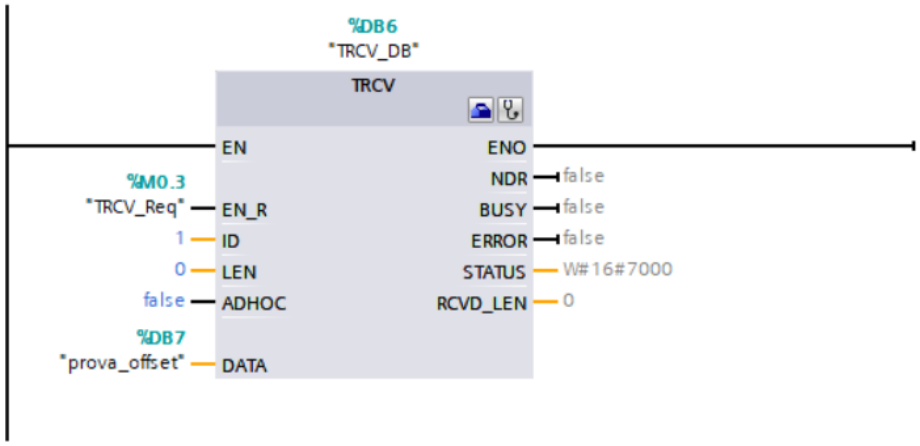



**Figure 3.20:** TRCV function the for TCP connection

### MATLAB

To set up the TCP connection from MATLAB first of all a connection object has to be created. The tcpip function is offered from MATLAB to this purpose. It takes as parameters the IP address of the partner, i.e. the PLC, the connection port set up in the configuration phase in Tia Portal as second parameter. The third and the fourth are used to define the role of Matlab in the netwrok that will be created. In our case, being the master in the TCP communication, it will operate as a server. The connection object is defined with the command written in Listing 4.1

**Listing 3.1:** Creation of the connection object

```
1 t = tcpip('192.168.0.1', 2000, 'NetworkRole', 'server');
```

Matlab offers four functions to manage the connection:

- **fopen()**. It opens the connection and requires the connection object as a parameter.
- **fclose()**. It closes the connection and requires the connection object as a parameter.
- **fwrite()**. It allows to send data from Matlab to the PLC through the previously established connection. It needs the connection object and the data that has to be sent as the two parameters.

- **fread()**. This function allows to read the data received through the connection. It has a required parameter, the connection object, and it is possible also to set the expected received data length as an optional parameter. It waits for the data and then return as output the read values when the expected length is reached or the connection timeout is expired.

#### Data blocks

Two data blocks were defined in Tia Portal:

- Sent_values, used to store the information about the actual position of the robot and send it to Matlab;
- Received_values, used to store the information about the movements that the robot has to perform in order to reach a point or grab/release an object.

Both the data blocks have the same structure, defined as follows:

1. Rotation, that stores an integer value corresponding to a certain counterclockwise rotation with respect to the initial position of the robot;
2. Vertical, that stores an integer value corresponding to a certain vertical movement towards the base of the robot with respect to the initial position of the robot;
3. Horizontal, that stores an integer value corresponding to a certain frontward movement with respect to the initial position of the robot;
4. Gripper, that stores a boolean value that is true when the gripper of the robot is fully closed or has to be closed, while it is false when the gripper of the robot is fully open or has to be open.

#### Matlab high level communication class, functions and scripts

Some high level communication functions and scripts were written in Matlab in order to facilitate the PLC programming for a specific task.

The script *Matlab_PLC_connection* creates a connection object and then open the connection listen for a client to connect. After the client connection, it initializes the *old_position* vector to the initial position of the robot.

Sent_values

| | Nome | Tipo di dati | Offset | Valore di avvio | A ritenzio... | Accessibile ... | Scrivi... | Visibile in .. | Valore di i.. | Commento |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Static | | | | | | | | | |
| 2 | Rotation | Int | 0.0 | 0 | | | | | | |
| 3 | Vertical | Int | 2.0 | 0 | | | | | | |
| 4 | Horizontal | Int | 4.0 | 0 | | | | | | |
| 5 | Gripper | Bool | 6.0 | false | | | | | | |

**(a)** Sent_values

Received_values

| | Nome | Tipo di dati | Offset | Valore di avvio | A ritenzio... | Accessibile ... | Scrivi... | Visibile in .. | Valore di i.. | Commento |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Static | | | | | | | | | |
| 2 | Rotation | Int | 0.0 | 0 | | | | | | |
| 3 | Vertical | Int | 2.0 | 0 | | | | | | |
| 4 | Horizontal | Int | 4.0 | 0 | | | | | | |
| 5 | Gripper | Bool | 6.0 | false | | | | | | |

**(b)** Received_values

**Figure 3.21:** Data Blocks

**Listing 3.2:** Matlab_PLC_connection.m

```
t=tcpip('192.168.0.1', 2000, 'NetworkRole', 'server');
fopen(t);
old_position = [0;0;0;0;0;0;0;0];
```

The script *reset_position* the connection buffer from remaining messages from the PLC and then reinitializes the vector *old_position* to the initial position of the robot. It is useful if the robot position is reset but the stop button has not been pressed, so the connection is still open.

**Listing 3.3:** reset_position.m

```
old_position = fread(t,8);
old_position = [0;0;0;0;0;0;0;0];
```

The function *move_robot()* was defined in order to move the robot: when the function is executed, it send a command to the PLC, that will move the robot, and then listen until the movement is completed and the PLC send back the position reached by the robot, which is written inside the *old_position* vector. If the robot is already in the desired position, the function terminates without doing nothing. The function has two required arguments:

- connection, that is the connection object to use:
- old_position, that is the old_position vector used to store the position of the robot.

This function has also a series of couple of optional arguments. In particular they are:

- the string 'rotation', followed by an integer which represent the a certain counterclockwise rotation with respect to the initial position of the robot. The initial position of

the robot is associated with a value of $-5$ degrees since, when the robot is in the reset position it is not perfectly aligned with the perpendicular direction to the side of the basis. The limit switch is pressed when the position of the robotic arm coincide with $-5$ degrees.

- the string 'vertical', followed by an integer value corresponding to a certain vertical movement towards the base of the robot with respect to the initial position of the robot;
- the string 'horizontal', followed by an integer value corresponding to a certain frontward movement with respect to the initial position of the robot;
- the string 'gripper', followed by a boolean value that is true when the gripper of the robot is fully closed or has to be closed, while it is false when the gripper of the robot is fully open or has to be open.

As it is possible to understand from the optional parameters definition, the positions that have to be passed are absolute positions with respect to the initial one. If some of the optional parameters are not used, the will be set by default to the previous one. The function returns the updated *old_position* vector, containing the reached position of the robot.

**Listing 3.4:** move_robot.m

```
1  function old_position = move_robot(connection, old_position,
       varargin)
2      connection.timeout = 180;
3      nreqargs = 2;
4      bit16rotation = [old_position(2,1) old_position(1,1)];
5      bit16vertical = [old_position(4,1) old_position(3,1)];
6      bit16horizontal = [old_position(6,1) old_position(5,1)];
7      gripper = old_position(7,1);
8
9      reached_rotation = typecast(uint8(bit16rotation),'uint16');
10     reached_vertical = typecast(uint8(bit16vertical),'uint16');
11     reached_horizontal = typecast(uint8(bit16horizontal),'uint16');
12
```

```
13      rotation = typecast(uint8(bit16rotation),'uint16');
14      vertical = typecast(uint8(bit16vertical),'uint16');
15      horizontal = typecast(uint8(bit16horizontal),'uint16');
16
17      if nargin > nreqargs
18          i=1;
19          while(i <= size(varargin,2))
20              switch lower(varargin{i})
21                  case  'rotation'
22                      temp_rotation = varargin{i+1};
23                      rotation = round((temp_rotation+5)*9.85);
24                      if rotation > 3300
25                          rotation = 3300;
26                      end
27                      if rotation < 0
28                          rotation = 0;
29                      end
30                      bit16rotation = typecast(uint16(rotation),'
                            uint8');
31                  case 'horizontal'
32                      temp_horizontal = varargin{i+1};
33                      horizontal = round(temp_horizontal*821);
34                      if horizontal > 78
35                          horizontal = 78;
36                      end
37                      if horizontal < 0
38                          horizontal = 0;
39                      end
40                      bit16horizontal = typecast(uint16(horizontal),'
                            uint8');
41                  case 'vertical'
```

```
42                      temp_vertical = varargin{i+1};
43                      vertical = round(temp_vertical*(-12857));
44
45                      if vertical > 1800
46                          vertical = 1800;
47                      end
48                      if vertical < 0
49                          vertical = 0;
50                      end
51                      bit16vertical = typecast(uint16(vertical),'
                            uint8');
52                  case 'gripper'
53                      gripper = varargin{i+1};
54              end
55              i= i+1;
56          end
57      end
58
59      codified_var = [bit16rotation(2); bit16rotation(1);
            bit16vertical(2); bit16vertical(1); bit16horizontal(2);
            bit16horizontal(1); gripper; 0];
60
61      if (isequal(gripper, old_position(7)) && isequal(
            reached_horizontal, horizontal) && (reached_rotation >=
            rotation - 10 && reached_rotation <= rotation + 10 ) && (
            reached_vertical >= vertical - 10 && reached_vertical <=
            vertical + 10))
62          return
63      end
64
65      fwrite(connection, codified_var)
```

```
66      old_position = fread(connection,8);
67  end
```

The function *emergency_reset* closes the connection, resets the *old_position* vector and open again the connection. It has the connection as required argument and returns the reinitialized *old_position* vector.

**Listing 3.5:** emergency_reset.m

```
1  function old_position = emergency_reset(connection)
2      fclose(connection);
3      old_position = [0;0;0;0;0;0;0];
4      fopen(connection);
```

Futhermore it was defined the class *gripper*, in order to make the use of the *move_robot* function more user-friendly. It has two properties:

- open, that is equal to false;
- close, that is equal to true.

**Listing 3.6:** gripper.m

```
1  classdef gripper
2    properties (Constant)
3        open = false;
4        close = true;
5    end
6  end
```

## 3.4 Ladder diagram

### 3.4.1 Variables table

The following variables were defined in order to program the PLC:

1. GripperOpenStop is a boolean that indicates that the gripper is fully open;

2. GripperCounter is a boolean that is used to increment a counter that represents how much the gripper is open or closed;
3. ArmBackStop is a boolean that indicates that the horizontal arm is completely set back;
4. ArmHCounter is a boolean that is used to increment a counter that represents how much the horizontal arm is forward or backward;
5. ArmTopStop is a boolean that indicates that the horizontal arm is completely set at the top;
6. ArmRotationStop is a boolean that indicates that the robot is fully clockwise rotated;
7. RotationalEncoder is an integer that indicates how much the robot is counterclockwise rotated and it is the output of an HSC (high speed counter);
8. VerticalEncoder is an integer that indicates how much the horizontal arm is moved down (high speed counter);
9. ConnectionButton is a boolean that is used to start the TCP connection with MATLAB;
10. StopButton is a boolean that indicates that an emergency occurred and so the robot has to be stopped;
11. ResetButton is a boolean that indicates that the robot position has to be reset to the initial one;
12. In3 is a boolean that represents an unused input of the PLC (it was defined for possible future extension of this work);
13. GripperOpen is a boolean that indicates that the gripper has to be open;
14. GripperClose is a boolean that indicates that the gripper has to be open;
15. ArmFront is a boolean that indicates that the horizontal arm has to be moved forward;
16. ArmBack is a boolean that indicates that the horizontal arm has to be moved back;
17. ArmDown is a boolean that indicates that the horizontal arm has to be moved down;
18. ArmUp is a boolean that indicates that the horizontal arm has to be moved up;

19. RotationClockwise is a boolean that indicates that the robot has to be rotated clockwise;
20. RotationCounterclockwise is a boolean that indicates that the robot has to be rotated counterclockwise;
21. GreenLed is a boolean that indicates that the green LED must be switched on;
22. RedLed is a boolean that indicates that the red LED must be switched on;
23. System_Byte is a byte that has to be activated from the properties of the PLC inside the clock and system merker tab (Figure3.23);
24. FirstScan is the first bit of the system byte and it is a boolean that is set to True when the PLC is switched on, then it is immediately set to False;
25. DiagStatusUpdate is the second bit of the system byte and it is not used in this project;
26. AlwaysTRUE is the third bit of the system byte and it is a boolean always set to True;
27. AlwaysFALSE is the fourth bit of the system byte and it is a boolean always set to False;
28. RotationalCounter is an integer used to indicate how much the robot is rotated counterclockwise with respect to the initial position;
29. Clock_Byte is a byte representing the system clock and has to be activated from the properties of the PLC inside the clock and system merker tab (Figure3.24);
30. Clock_10Hz is the first bit of the clock byte and it is set to True and immediately set to False at the frequency of 10 Hz;
31. Clock_5Hz is the second bit of the clock byte and it is set to True and immediately set to False at the frequency of 5 Hz;
32. Clock_2.5Hz is the third bit of the clock byte and it is set to True and immediately set to False at the frequency of 2.5 Hz;
33. Clock_2Hz is the fourth bit of the clock byte and it is set to True and immediately set to False at the frequency of 2 Hz;
34. Clock_1.25Hz is the fifth bit of the clock byte and it is set to True and immediately set to False at the frequency of 1.25 Hz;

35. Clock_1Hz is the sixth bit of the clock byte and it is set to True and immediately set to False at the frequency of 1 Hz;

36. Clock_0.625Hz is the seventh bit of the clock byte and it is set to True and immediately set to False at the frequency of 0.625 Hz;

37. Clock_0.5Hz is the eighth bit of the clock byte and it is set to True and immediately set to False at the frequency of 0.5 Hz;

38. VerticalCounter is an integer used to indicate how much the horizontal arm is moved down with respect to the initial position;

39. ForceReset is a boolean that is set to True during the reset of the robot to the initial position;

40. ResetCompleted is a boolean that is set to True when the reset operation is completed;

41. TCON_Req is a boolean that was used during the environment setup phase to establish the TCP connection with MATLAB;

42. TDIS_Req is a boolean that was used during the environment setup phase to close the TCP connection with MATLAB;

43. TSEND_Req is a boolean that was used during the environment setup phase to write inside the TCP connection send buffer;

44. TRCV_Req is a boolean that was used during the environment setup phase to read the TCP connection receive buffer;

45. VerticalOverCounter is an integer that represents the maximum of the confidence interval for the vertical movement;

46. VerticalUnderCounter is an integer that represents the minimum of the confidence interval for the vertical movement;

47. RotationOverCounter is an integer that represents the maximum of the confidence interval for the rotation movement;

48. RotationUnderCounter is an integer that represents the minimum of the confidence interval for the rotation movement;

49. NormalizedRotationUnderCounter is an integer that is equal to the RotationUnderCounter value, with the exception that, when RotationUnderCounter is negative, it is zero;

50. NormalizedVerticalUnderCounter is an integer that is equal to the VerticalUnderCounter value, with the exception that, when VerticalUnderCounter is negative, it is zero;

51. StopAll is a boolean that is True when the emergency stop button is pressed;

52. HRegistered is a boolean that indicates that the horizontal position of the robot has been written into the Sent_values data block;

53. VRegistered is a boolean that indicates that the vertical position of the robot has been written into the Sent_values data block;

54. RRegistered is a boolean that indicates that the rotational position of the robot has been written into the Sent_values data block.

Tabella delle variabili standard

| | Nome | Tipo di dati | Indirizzo | Ritenz... | Acces... | Scrivi... | Visibil... |
|---|---|---|---|---|---|---|---|
| 1 | GripperOpenStop | Bool | %I0.0 | ☐ | ☑ | ☑ | ☑ |
| 2 | GripperCounter | Bool | %I0.1 | ☐ | ☑ | ☑ | ☑ |
| 3 | ArmBackStop | Bool | %I0.2 | ☐ | ☑ | ☑ | ☑ |
| 4 | ArmHCounter | Bool | %I0.3 | ☐ | ☑ | ☑ | ☑ |
| 5 | ArmTopStop | Bool | %I0.4 | ☐ | ☑ | ☑ | ☑ |
| 6 | ArmRotationStop | Bool | %I0.5 | ☐ | ☑ | ☑ | ☑ |
| 7 | RotationalEncoder | DInt | %ID1000 | ☐ | ☑ | ☑ | ☑ |
| 8 | VerticalEncoder | DInt | %ID1004 | ☐ | ☑ | ☑ | ☑ |
| 9 | ConnectionButton | Bool | %I1.2 | ☐ | ☑ | ☑ | ☑ |
| 10 | StopButton | Bool | %I1.3 | ☐ | ☑ | ☑ | ☑ |
| 11 | ResetButton | Bool | %I1.4 | ☐ | ☑ | ☑ | ☑ |
| 12 | In3 | Bool | %I1.5 | ☐ | ☑ | ☑ | ☑ |
| 13 | GripperOpen | Bool | %Q0.0 | ☐ | ☑ | ☑ | ☑ |
| 14 | GripperClose | Bool | %Q0.1 | ☐ | ☑ | ☑ | ☑ |
| 15 | ArmFront | Bool | %Q0.2 | ☐ | ☑ | ☑ | ☑ |
| 16 | ArmBack | Bool | %Q0.3 | ☐ | ☑ | ☑ | ☑ |
| 17 | ArmDown | Bool | %Q0.4 | ☐ | ☑ | ☑ | ☑ |
| 18 | ArmUp | Bool | %Q0.5 | ☐ | ☑ | ☑ | ☑ |
| 19 | RotationClockwise | Bool | %Q0.6 | ☐ | ☑ | ☑ | ☑ |
| 20 | RotationCounterclockwise | Bool | %Q0.7 | ☐ | ☑ | ☑ | ☑ |
| 21 | GreenLed | Bool | %Q1.0 | ☐ | ☑ | ☑ | ☑ |
| 22 | RedLed | Bool | %Q1.1 | ☐ | ☑ | ☑ | ☑ |
| 23 | System_Byte | Byte | %MB6 | ☐ | ☑ | ☑ | ☑ |
| 24 | FirstScan | Bool | %M6.0 | ☐ | ☑ | ☑ | ☑ |
| 25 | DiagStatusUpdate | Bool | %M6.1 | ☐ | ☑ | ☑ | ☑ |
| 26 | AlwaysTRUE | Bool | %M6.2 | ☐ | ☑ | ☑ | ☑ |
| 27 | AlwaysFALSE | Bool | %M6.3 | ☐ | ☑ | ☑ | ☑ |

**(a)** Variables table (1)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 28 | RotationalCounter | UInt | %MW2 | ☐ | ☑ | ☑ | ☑ |
| 29 | Clock_Byte | Byte | %MB0 | ☐ | ☑ | ☑ | ☑ |
| 30 | Clock_10Hz | Bool | %M0.0 | ☐ | ☑ | ☑ | ☑ |
| 31 | Clock_5Hz | Bool | %M0.1 | ☐ | ☑ | ☑ | ☑ |
| 32 | Clock_2.5Hz | Bool | %M0.2 | ☐ | ☑ | ☑ | ☑ |
| 33 | Clock_2Hz | Bool | %M0.3 | ☐ | ☑ | ☑ | ☑ |
| 34 | Clock_1.25Hz | Bool | %M0.4 | ☐ | ☑ | ☑ | ☑ |
| 35 | Clock_1Hz | Bool | %M0.5 | ☐ | ☑ | ☑ | ☑ |
| 36 | Clock_0.625Hz | Bool | %M0.6 | ☐ | ☑ | ☑ | ☑ |
| 37 | Clock_0.5Hz | Bool | %M0.7 | ☐ | ☑ | ☑ | ☑ |
| 38 | VerticalCounter | UInt | %MW4 | ☐ | ☑ | ☑ | ☑ |
| 39 | ForceReset | Bool | %M7.0 | ☐ | ☑ | ☑ | ☑ |
| 40 | ResetCompleted | Bool | %M7.1 | ☐ | ☑ | ☑ | ☑ |
| 41 | TCON_Req | Bool | %M1.0 | ☐ | ☑ | ☑ | ☑ |
| 42 | TDIS_Req | Bool | %M1.1 | ☐ | ☑ | ☑ | ☑ |
| 43 | TSEND_Req | Bool | %M1.2 | ☐ | ☑ | ☑ | ☑ |
| 44 | TRCV_Req | Bool | %M1.3 | ☐ | ☑ | ☑ | ☑ |
| 45 | VerticalOverCounter | Int | %MW8 | ☐ | ☑ | ☑ | ☑ |
| 46 | VerticalUnderCounter | Int | %MW10 | ☐ | ☑ | ☑ | ☑ |
| 47 | RotationOverCounter | Int | %MW12 | ☐ | ☑ | ☑ | ☑ |
| 48 | RotationUnderCounter | Int | %MW14 | ☐ | ☑ | ☑ | ☑ |
| 49 | NormalizedRotationUnderCoun... | Int | %MW16 | ☐ | ☑ | ☑ | ☑ |
| 50 | NormalizedVerticalUnderCounte | Int | %MW18 | ☐ | ☑ | ☑ | ☑ |
| 51 | StopAll | Bool | %M20.1 | ☐ | ☑ | ☑ | ☑ |
| 52 | HRegistered | Bool | %M20.2 | ☐ | ☑ | ☑ | ☑ |
| 53 | RRegistered | Bool | %M20.3 | ☐ | ☑ | ☑ | ☑ |
| 54 | VRegistered | Bool | %M20.4 | ☐ | ☑ | ☑ | ☑ |

**(b)** Variables table (2)

**Figure 3.22:** Variables table

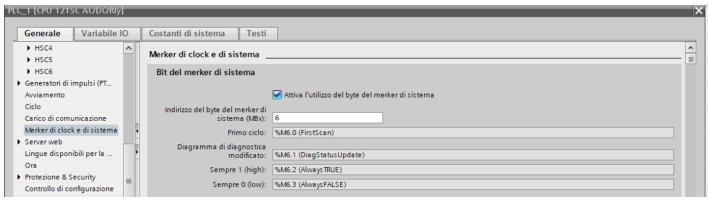



**Figure 3.23:** System byte activation

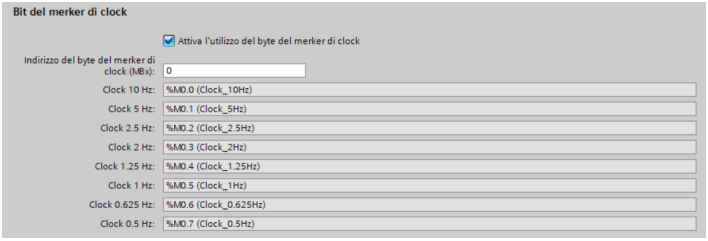



**Figure 3.24:** Clock merkers activation

### 3.4.2 Ladder diagram

The first operation performed when the PLC is turned on is the reset of the robot to its reference initial position.

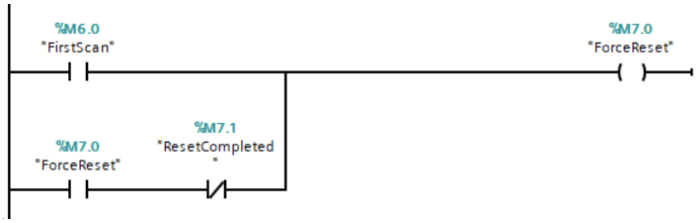



**Figure 3.25:** Position initialization

As it is possible to see in Figure3.25, the ForceReset variable is set to True on the startup and then it is self-retained until the reset operation is completed (when the ResetCompleted variable is set to True).

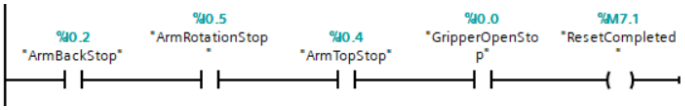



**Figure 3.26:** Reset completed

The ResetCompleted variable is set when all the limit switches are pressed (Figure3.26).

When the reset button is pressed and hold for 2 seconds, the force reset variable is set and than it is self-retained until the reset is completed (Figure3.27).

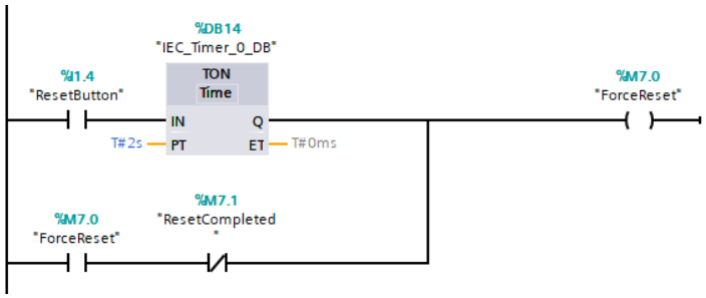



**Figure 3.27:** Reset button

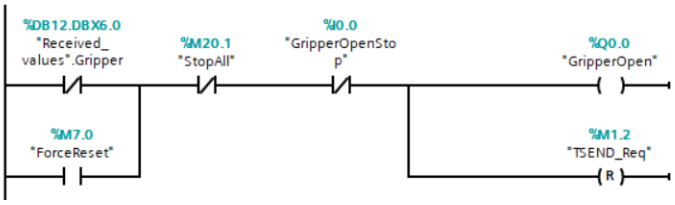



**Figure 3.28:** Gripper open

As it is possible to see in Figure3.28, the gripper is open when the corresponding value received from MATLAB is False or during the reset operation. The movement continues until the corresponding limit switch is pressed or the emergency stop is activated. During the opening of the gripper TSEND_Req is reset because it has to be set when all the movements are completed.

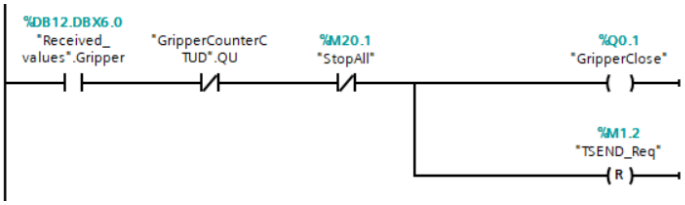



**Figure 3.29:** Gripper close

As it is possible to see in Figure3.29, the gripper is closed when the corresponding value received from MATLAB is True and the emergency stop has not been activated. The movement continues until the motor is forced three times. During the closing of the gripper TSEND_Req is reset because it has to be set when all the movements are completed.

As it is possible to see in Figure3.30, the gripper counter is increased during the closure of the gripper with a clock of 10 Hz, which corresponds to the gripper pulse counter frequency. The gripper counter is decreased each time the gripper pulse counter is set during the closure of the gripper. In this way, it is increased and decreased simultaneously. However, if the motor

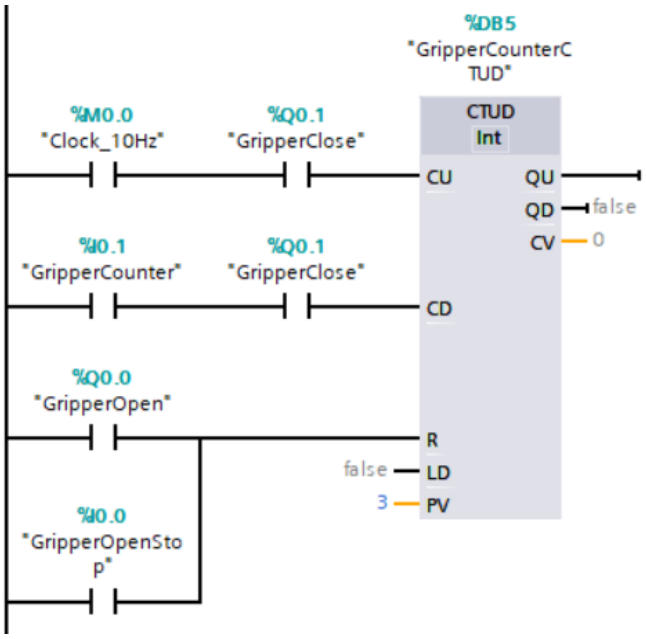



**Figure 3.30:** Gripper counter

is forced and thus the pulse counter is not switched on, the gripper counter is only increased. The gripper counter is reset when the gripper is opening or when the corresponding limit switch is pressed.

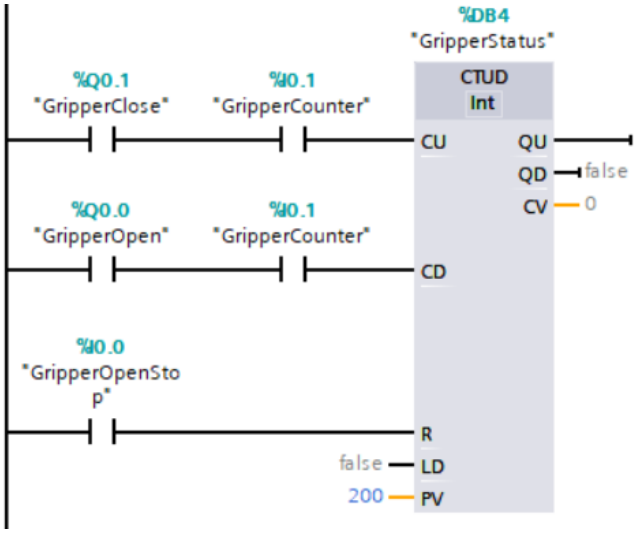



**Figure 3.31:** Gripper status

As it is possible to see in Figure3.31, the gripper status counter is increased during the opening of the gripper each time the gripper pulse counter is set. The gripper status counter is decreased during the closure of the gripper each time the gripper pulse counter is set. In this way, it is possible to store the information how the gripper is open or closed. The gripper

status counter is reset when the corresponding limit switch is pressed.

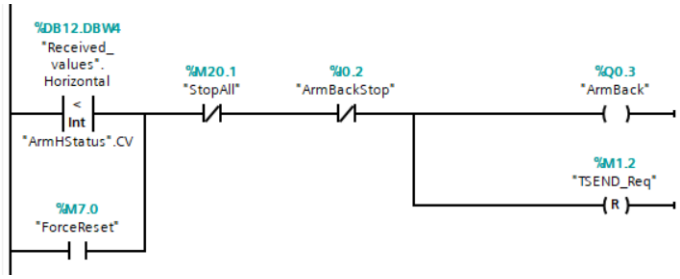



**Figure 3.32:** Arm back

As it is possible to see in Figure3.32, the horizontal arm of the robot is moved backward when the corresponding value received from MATLAB is less than the ArmHStatus counter value or during the reset operation. In both cases the movement is stopped if the emergency stop is activated of if the corresponding limit switch is pressed. During the movement of the arm TSEND_Req is reset because it has to be set when all the movements are completed.

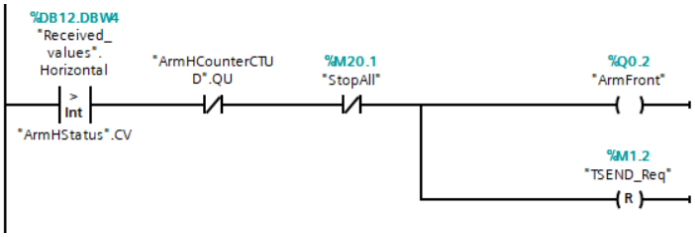



**Figure 3.33:** Arm front

As it is possible to see in Figure3.33, the horizontal arm of the robot is moved forward when the corresponding value received from MATLAB is greater than ArmHStatus counter value and the emergency stop has not been activated. The movement is stopped if the motor is forced three times. During the movement of the arm TSEND_Req is reset because it has to be set when all the movements are completed.

As it is possible to see in Figure3.34, the horizontal movement counter is increased during the forward movement of the horizontal arm with a clock of 10 Hz, which corresponds to the corresponding pulse counter frequency. The horizontal movement counter is decreased each time the corresponding pulse counter is set during the forward movement. In this way, it is increased and decreased simultaneously. However, if the motor is forced and thus the pulse counter is not switched on, the horizontal movement counter is only increased. The horizontal movement counter is reset during the backward movement or when the corresponding limit

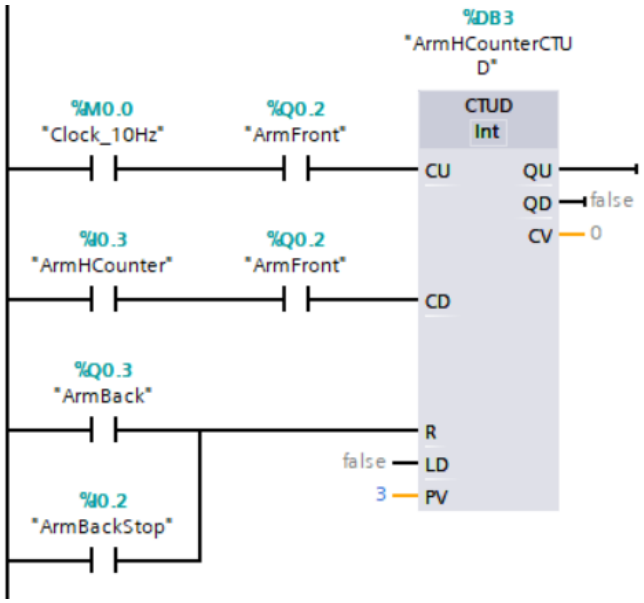



**Figure 3.34:** Arm horizontal movement counter

switch is pressed.

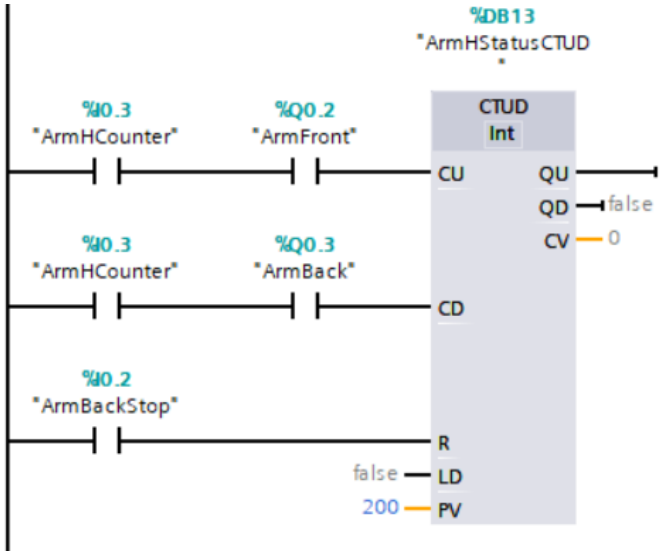



**Figure 3.35:** Arm horizontal movement status

As it is possible to see in Figure3.35, the arm horizontal movement status counter is increased during the forward movement each time the corresponding pulse counter is set. The arm horizontal movement status counter is decreased during the backward movement each time the corresponding pulse counter is set. In this way, it is possible to store the information about the horizontal position of the arm. The arm is reset when the corresponding limit switch is pressed.

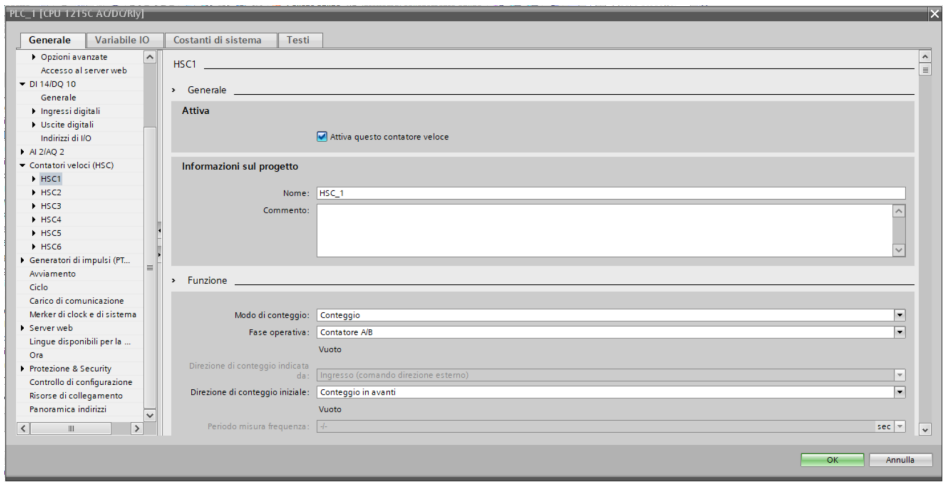
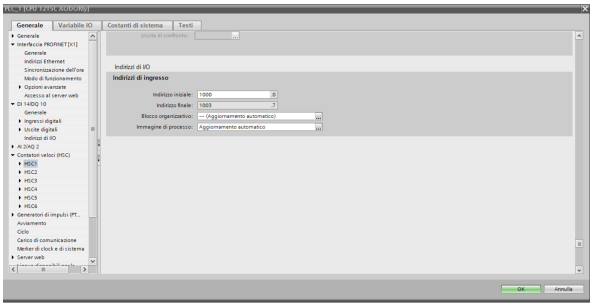
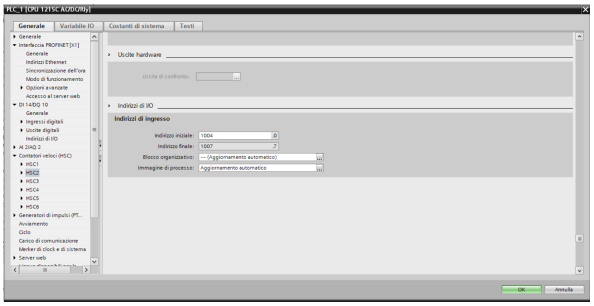



**Figure 3.36:** HSC activation and configuration

The vertical and rotational movements are monitored using the two encoders. Two HSCs (high speed counters) were activated to read values from the encoder from the PLC properties inside the HSC tab and were set to A/B count mode (Figure3.36). In this way the HSCs value is incremented or decremented by 1 depending on the time order of the rising edges of the corresponding encoder signals (A and B).



**(a)** Horizontal encoder addresses



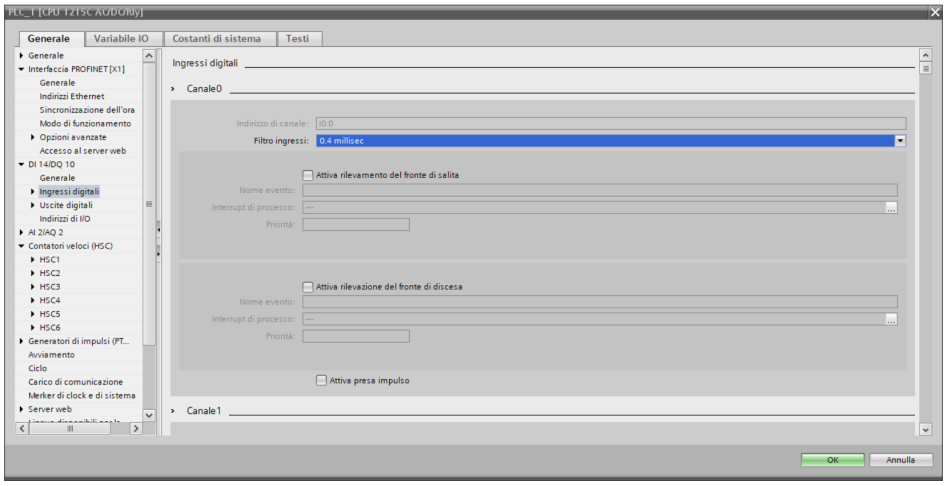
**(b)** Vertical encoder addresses

**Figure 3.37:** HSCs values addresses configuration

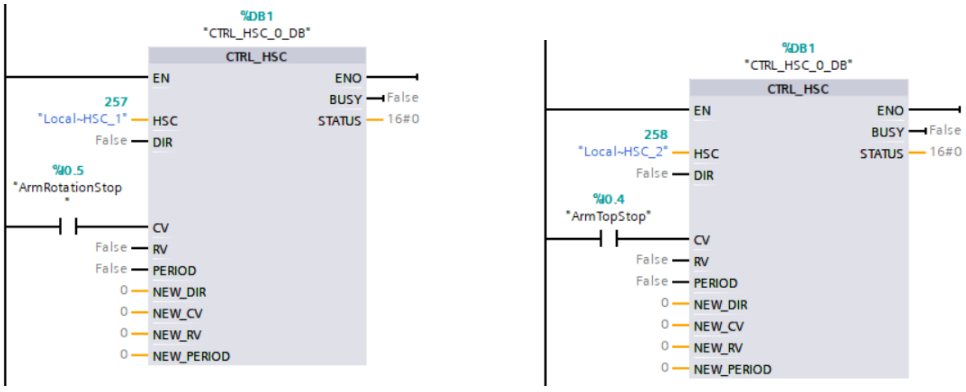
The rotational and vertical HSCs were configured to store the counter valued respectively in the memory addresses of the variables RotationalEncoder and VerticalEncoder (Figure3.37).

The input frequency for the HSC was set to 2.5 kHz (Figure 3.38), because the maximum freuqency of the encoders is 1 kHz and accordingly to the Nyquist theorem the sampling frequency should be greater or equal to the double of the maximum frequency of the signal.

**Figure 3.38:** Encoders frequency




]



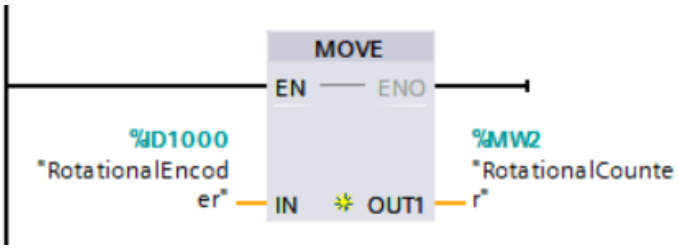

**(a)** HSC 1 **(b)** HSC 2

**Figure 3.39:** HSCs

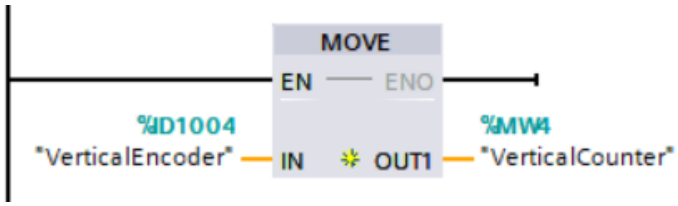
The rotation HSC (Figure3.39a) and the vertical movement HSC (Figure3.39b) are stopped when the corresponding limit switch is pressed.

The values stored in the variables corresponding to the two HSCs values were copied inside the variables RotationalCounter and VerticalCounter (Figure3.40). This was done because it is a good practice to avoid to use directly the variables in which the values counted by the HSCs are stored: in fact in this way it is impossible that those values are erroneously modified.

The increment on subsequent values that were possible to read from the encoders was not equal to 1 unit. Therefore they were defined two confidence intervals (one for each

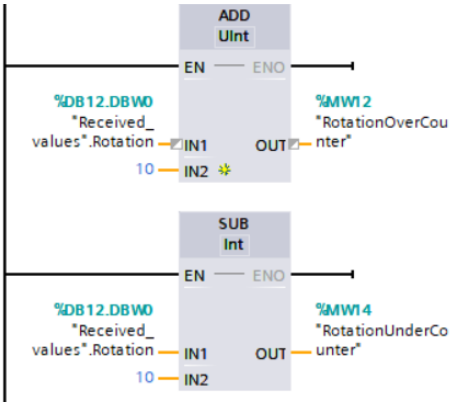



**(a)** RotationalEncoder variable copy

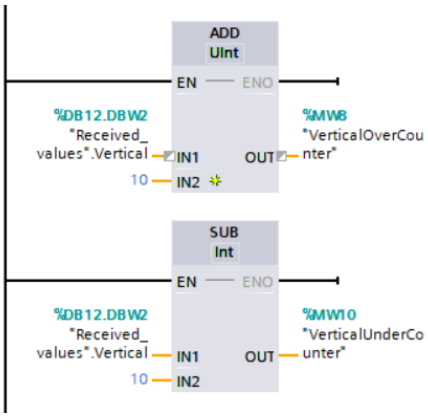



**(b)** VerticalEncoder variable copy

**Figure 3.40:** HSCs variables copy




**(a)** Rotation confidence interval




**(b)** Vertical confidence interval

**Figure 3.41:** Confidence intervals

encoder), within which the position error is negligible (in fact the difference cannot be noticed by a human eye). The intervals are 20 units large and are centered in the corresponding desired positions. The maximum and the minimum of each interval computation is shown in Figure3.41.

It may happen that the minimum of the confidence intervals is a negative value. When this happens, the minimum is set to 0 by using two MAX blocks in which each of the computed minimums are compared with 0 and the maximum between the compared values is then stored in a new variable called NormalizedRotationUnderCounter for the rotation and NormalizedVerticalUnderCounter for the vertical movement (Figure3.42).

As it is possible to see in Figure3.43, the horizontal arm of the robot is moved up when

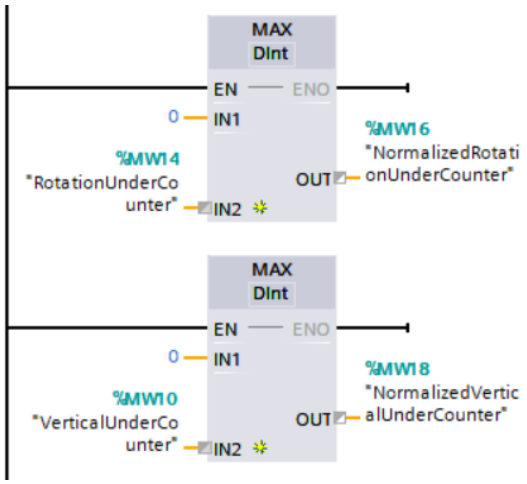



**Figure 3.42:** Confidence intervals minimum adjustment

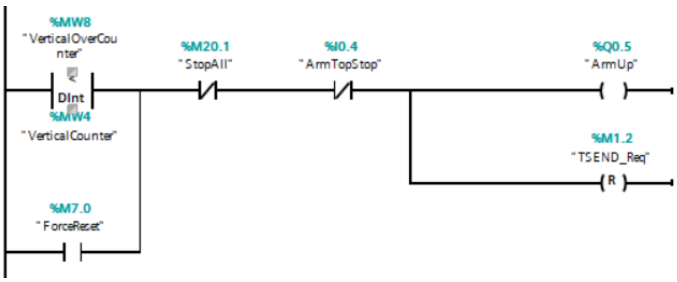



**Figure 3.43:** Arm up

the minimum of the corresponding confidence interval is less than the value read from the corresponding encoder or during the reset operation. In both cases the movement continues until the corresponding limit switch is pressed or the emergency stop is activated. During the movement of the arm TSEND_Req is reset because it has to be set when all the movements are completed.

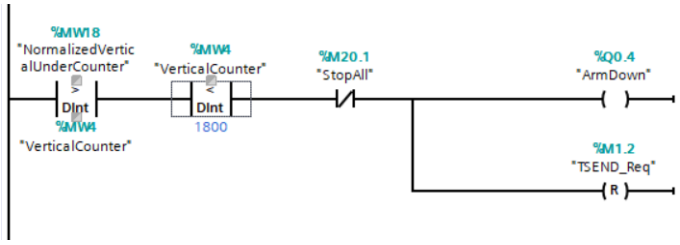



**Figure 3.44:** Arm down

As it is possible to see in Figure3.44, the horizontal arm of the robot is moved down when the maximum of the corresponding confidence interval is greater than the value read from the corresponding encoder. The movement is stopped if the emergency stop is activated or

the maximum reachable point is reached. During the movement of the arm TSEND_Req is reset because it has to be set when all the movements are completed.

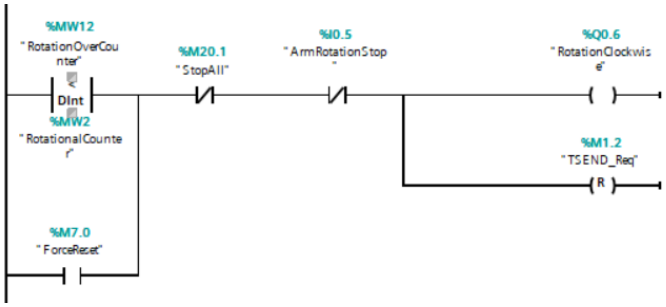



**Figure 3.45:** Rotation clockwise

As it is possible to see in Figure3.45, the robot is rotated clockwise when the maximum of the corresponding confidence interval is less than the value read from the corresponding encoder or during the reset operation. In both cases the movement continues until the corresponding limit switch is pressed or the emergency stop is activated. During the rotation of the robot TSEND_Req is reset because it has to be set when all the movements are completed.

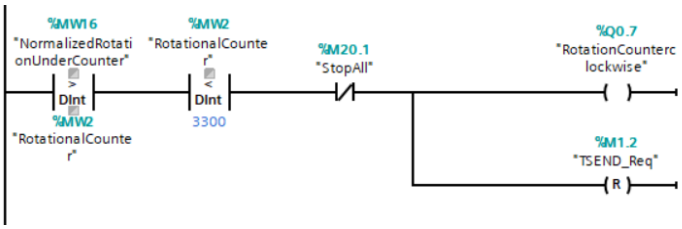
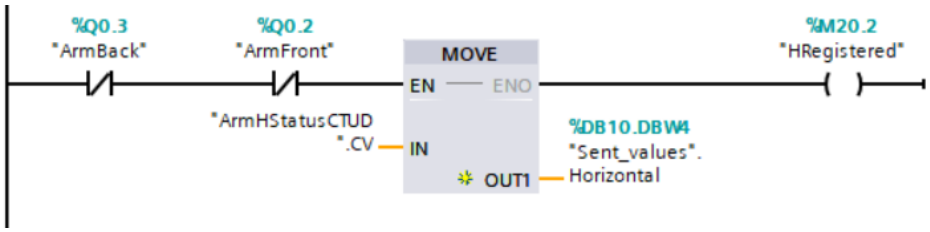



**Figure 3.46:** Rotation counterclockwise

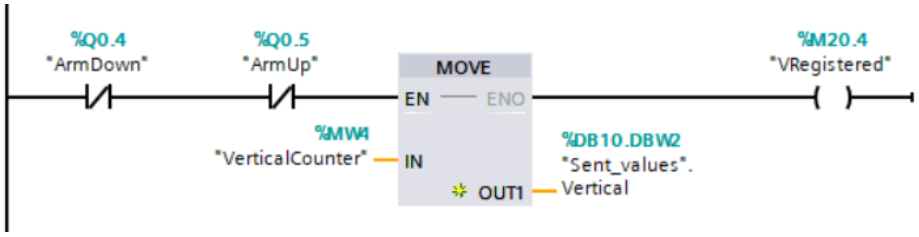
As it is possible to see in Figure3.46, the robot is rotated counterclockwise when the minimum of the corresponding confidence interval is greater than the value read from the corresponding encoder. The movement is stopped if the emergency stop is activated or the maximum reachable point is reached. During the rotation TSEND_Req is reset because it has to be set when all the movements are completed.

As it is possible to see in Figure3.47 the values of the variables stored into the Sent_values data block are updated each time the corresponding movement has ended.

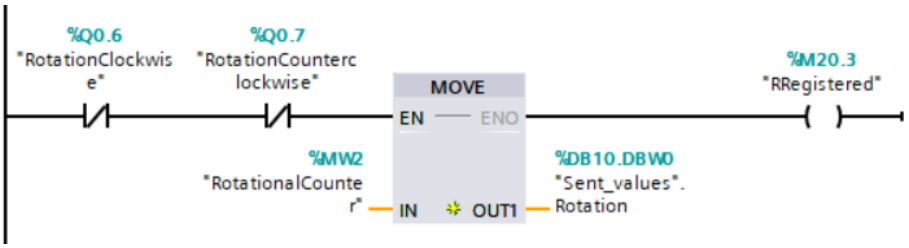
As it is possible to see in Figure3.48, when all the movements are completed and the corresponding reached position has been written into the Sent_values data block, after one second (the delay ensure that the write operations are finished before to send the data), TSEND_Req is set and thus these values are sent to MATLAB.



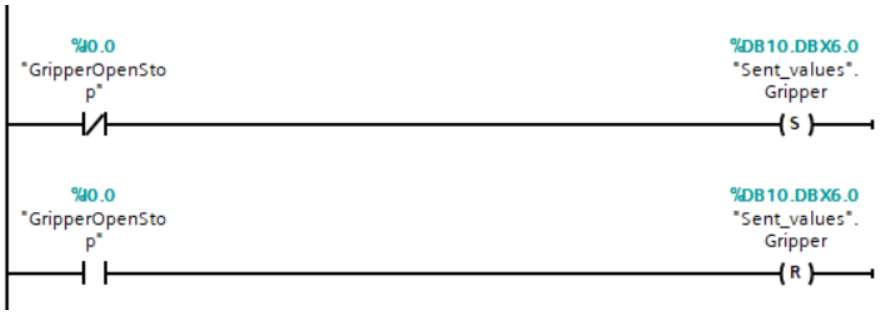

**(a)** Horizontal value update in Sent_values




**(b)** Vertical value update in Sent_values




**(c)** Rotation value update in Sent_values




**(d)** Gripper value update in Sent_values

**Figure 3.47:** Sent_values update

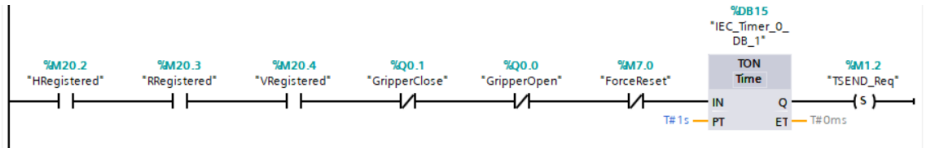
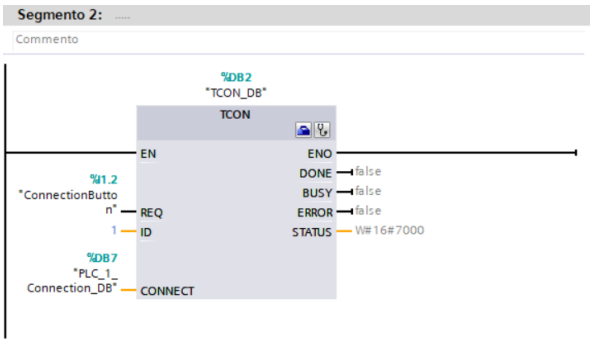
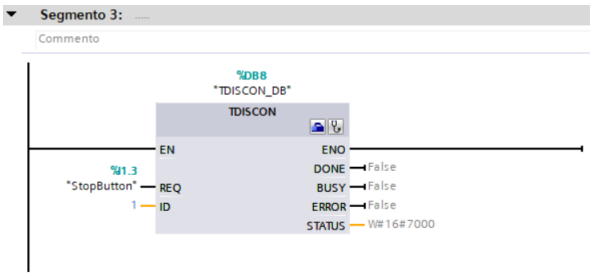



**Figure 3.48:** Send values to MATLAB

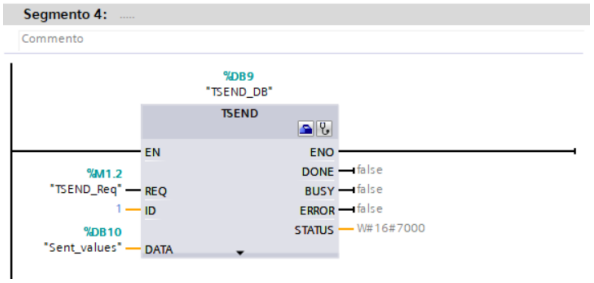
As it is possible to see in Figure3.49a the PLC connection with Matlab is open on the PLC side when the connection button is pressed. From Figure3.49b it is possible to see that when the emergency stop button is pressed the connection is closed on the PLC side. As it is possible to see in Figure3.49c, when TSEND_Req is set, the values stored in Sent_values



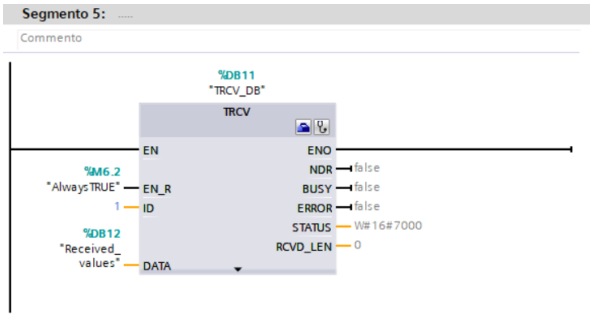

**(a)** Connection block




**(b)** Disconnection block




**(c)** Send block




**(d)** Receive block

**Figure 3.49:** TCP Connection blocks

data block are sent to Matlab. From Figure3.49d it is possible to see that the Matlab and those values are then stored into the Received_values data block.

As it is possible to see in Figure3.50, when the stop button is pressed, the emergency stop is activated and then it is self-retained.

As it is possible to see in Figure3.51, the pressure of the emergency stop button reset the


**Figure 3.50:** Stop button

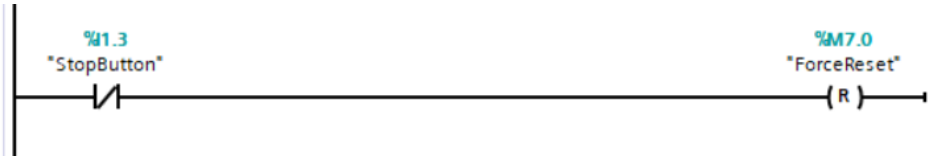



**Figure 3.51:** Emergency stop during reset

ForceReset variable, thus it is possible to stop the robot also during the reset operation.

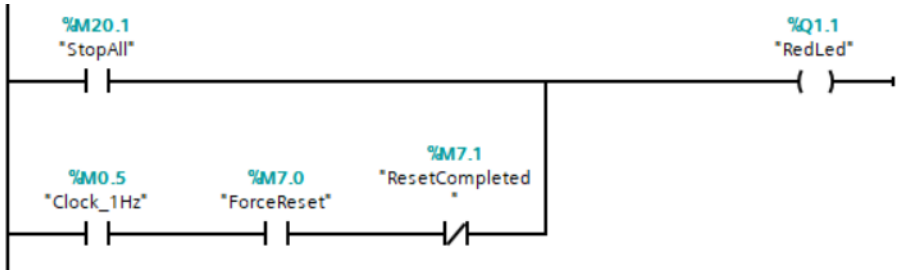



**Figure 3.52:** Red LED

As it is possible to see in Figure3.52, when the emergency stop is active the red LED is switched on, while during the reset operation it blinks with a frequency of 1Hz.

As it is possible to see in Figure3.53, the green LED is switched on when the robot is not moving and it has not been stopped using the emergency stop button, while it blinks when the robot is moving.

As it is possible to see in Figure3.54, during the reset operation all the variables inside the Sent_values and Received_values data blocks are imposed equal to 0 or false depending on their type (integer or boolean), the emergency stop is reset and the green LED is switched off until the reset operation is completed.

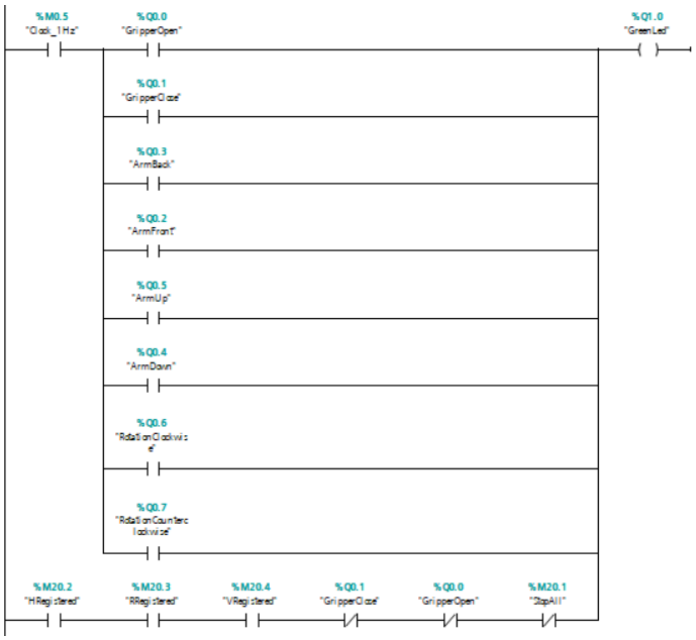



**Figure 3.53:** Green LED

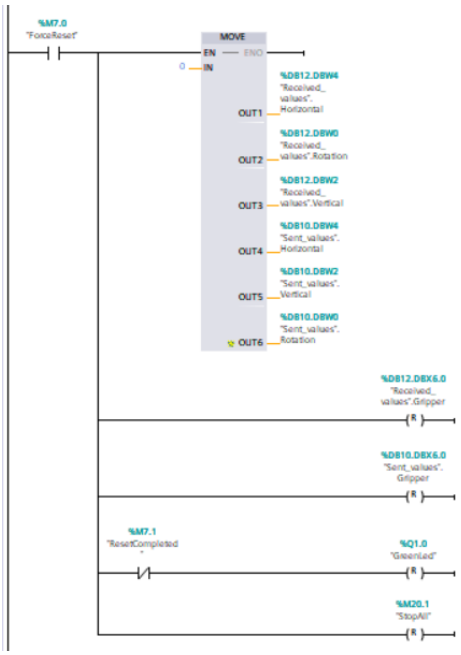



**Figure 3.54:** Reset operations

## 3.5 CoppeliaSim model

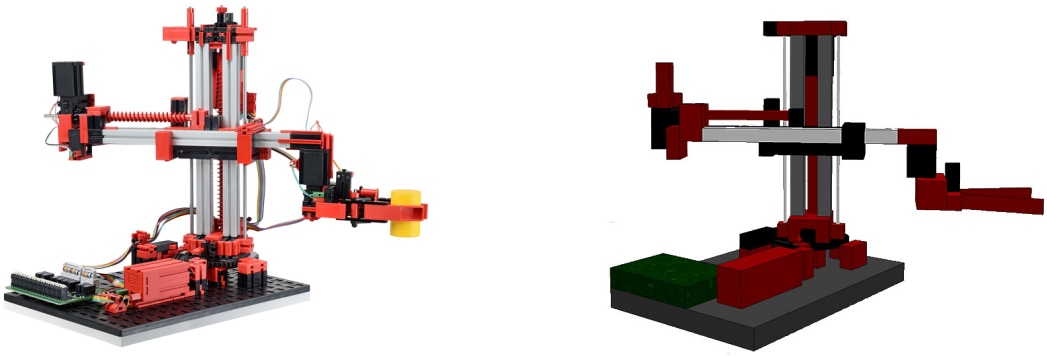
The model of the robotic arm has been created through the version 4.1.0 of the software CoppeliaSim (formerly known as *V-REP*). The goal of the model realization process is to create a environment which is able to perform simulations of the works that will be done by

the robot. The approach of the simulations enable the project developers to test in a secure sandbox all the movement which the robot needs to perform, and allow (if it is well designed) to understand which are the constraints in the moves during the scene performance. To accomplish this task, the model was realized in order to work in two different ways:

- a simulation in which the robot can reach all the points through **direct** kinematics, in which the movements are imposed by passing the values to set for each joint;
- a simulation in which the robot can reach all the points through **inverse** kinematics, in which the movements are imposed by passing directly the point the robot's tip need to reach.

The process of modeling the robot requires an analysis of the pieces which need to be realized in the model in order to reach a good level of approximation.

In this specific project, the model, which is represented in Figure 3.58b, is composed by two main elements which represent the base on which the robot lies and the robot itself. Moreover, on the base element, all the electronic components which enable motors and sensors are modeled to represent obstacles for the arm movements, while on the robot components, all the joins, links and support structures are implemented.



**(a)** Real Robot **(b)** 3D Model

**Figure 3.55:** Comparative between real robot and 3D model

No 3D model was provided by the manufacturer of the robot and, in particular, on the website of the *Fischertecnick robotic arm*[1], only a series of images, videos and datasheets are provided. For this reason, not having a complete drawing of the robot with quotas and measures, a meticulous process of measurement was necessary and it strictly followed the one of modeling. All the elements inside the model are realized by only using a set of primitive

shapes (i.e., cuboids and cylinders), joints (i.e., revolute and prismatic ones) and sensors (the proximity ones). More in details, all the moving parts of the robot with except of the gripper (which is a static part), are realized as moving part of the 3D model. With regard to the gripper instead, it could be implemented a new version which enable the grip ability in future works. Inspecting the model one revolute joint (which enable the robot to rotate) and two prismatic joints (one which enable the robot to move vertically and the other one for the horizontal movements) can be found. As for the physics linked to the real model, each movement for the 3D model robot has a spatial range:

- the revolute joint can assume position in the interval $[-5.00, 330.00]$, with a range of 335°;
- the prismatic joint for vertical movements can assume position in the interval $[-14.00, 0.00]$, with a range of 14.00 cm;
- the prismatic joint for horizontal movements can assume position in the interval $[0.00, 9.50]$, with a range of 9.50 cm.

More specifically, to better simulate the real behaviour of the robot, all the *end-of-stroke* buttons are implemented through proximity sensors whose, linked to a *Non threaded* child script, enable the robot to stop exactly when the end-of-stroke buttons would be pressed in real world. In the used proximity sensors, which are of type *ultrasonic*, the property “Explicit handling" is enabled (to allow the control through the script), the used “Volume parameters" are listed in , as shown in Figure 3.56.

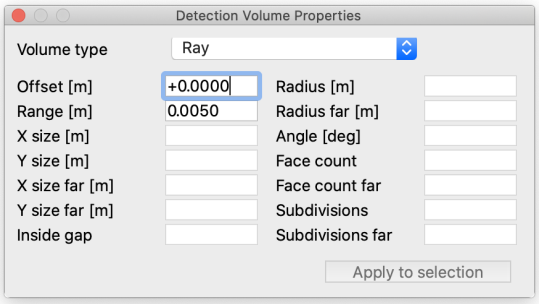
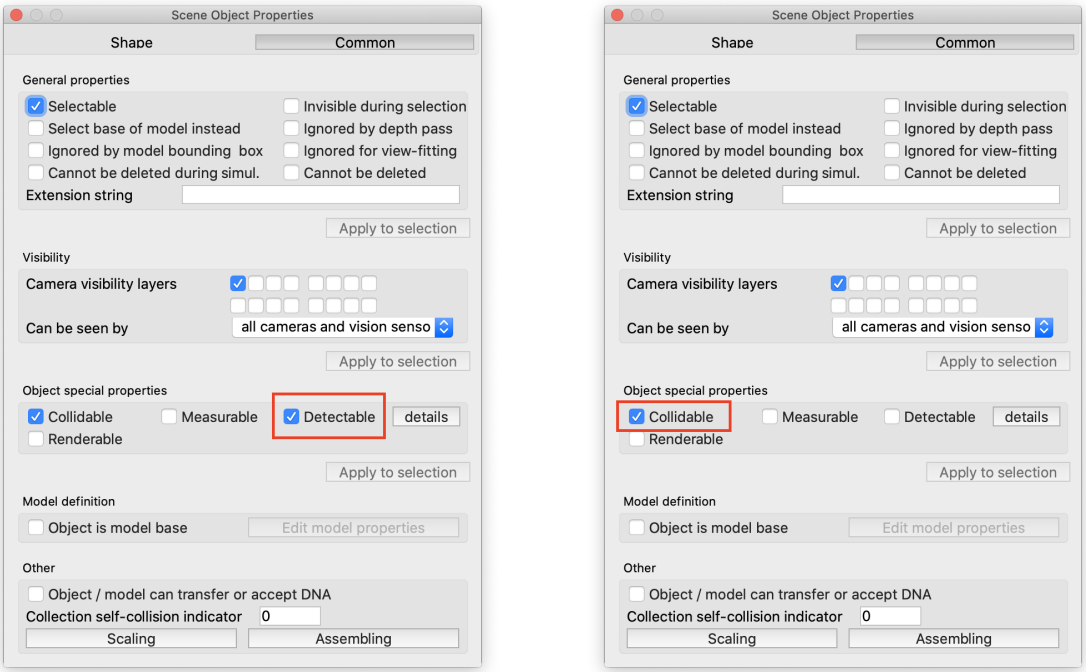



**Figure 3.56:** Volume parameters

The sensors are enabled on both side of each link which can be moved and they are “sensitive" only to the particular joint that will be able to approach them. To accomplish this

specific task, all the links, which have to be sensed by the sensors, have to be configured as **Detectable** objects, as shown in Figure 3.57a.




**(a)** Detectable object **(b)** Collidable object

**Figure 3.57:** Setup for object

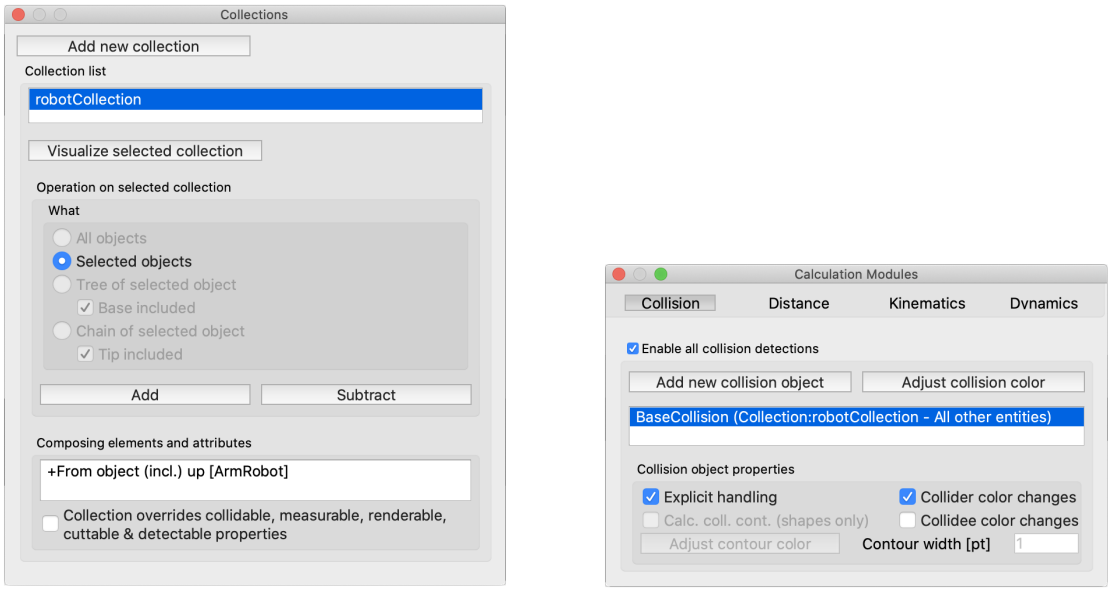
Furthermore, in the realized scene, it is also configured a system to detect collision. More specifically, a collection is configured which contains all the “moving elements" of the robot, as shown in the Figure 3.58a. It is also needed to add a new object collision in the “Calculation Modules" window as shown in Figure 3.58b, reached from the voice “Tools" in the menubar. Next step requires to set all the elements which can collide. To do that, object needs to be a “Collidable Object", as shown in Figure 3.57b.

Listing 3.7 display the lua code used to manage the sensors and the collisions.

In particular the content of the *sysCall_init()* is used to retrieve the handle for the object which are used in the other functions. In the *sysCall_actuation()* the code is written to make movements up to when the joints are not pressing the end-of-stroke buttons. Inside the *sysCall_sensing()* collision checks are done and sensing is performed through the sensors.

**Listing 3.7:** Non threaded child script for register collision

```
1 function sysCall_init()
2     revoluteJoint = sim.getObjectHandle('RevoluteJoint')
```




**(a)** Robot collection **(b)** Calculation modules

**Figure 3.58:** Setup for collision

```
3  prismaticVerticalJoint = sim.getObjectHandle('
       PrismaticVerticalJoint')
4  prismaticHorizontalJoint = sim.getObjectHandle('
       PrismaticHorizontalJoint')
5  rotationStop0 = sim.getObjectHandle('RotationStop0')
6  rotationStop1 = sim.getObjectHandle('RotationStop1')
7  verticalStop0 = sim.getObjectHandle('VerticalStop0')
8  verticalStop1 = sim.getObjectHandle('VerticalStop1')
9  horizontalStop0 = sim.getObjectHandle('HorizontalStop0')
10 horizontalStop1 = sim.getObjectHandle('HorizontalStop1')
11 verticalTriggerTopStop = sim.getObjectHandle('ExtraStuff13')
12 verticalTriggerBottomStop = sim.getObjectHandle('
       ConnectionToVerticalSupport1')
13 rotationTriggerStop = sim.getObjectHandle('
       ExtraStuffForRotation')
14 horizontalTriggerBackStop = sim.getObjectHandle('
       ExtraStuffBackStop')
15 horizontalTriggerFrontStop = sim.getObjectHandle('
```

```
        ExtraStuffFrontStop')
16 end
17
18 function sysCall_actuation()
19     if(r0 == 1 or r1 == 1) then
20         stopRotationPosition = sim.getJointPosition(revoluteJoint)
21         sim.setJointTargetPosition(revoluteJoint,
               stopRotationPosition)
22     else
23         sim.setJointTargetPosition(revoluteJoint, 305*math.pi/180)
24     end
25     if(v0 == 1 or v1 == 1) then
26         stopVerticalPosition = sim.getJointPosition(
               prismaticVerticalJoint)
27         sim.setJointTargetPosition(prismaticVerticalJoint,
               stopVerticalPosition)
28     else
29         sim.setJointTargetPosition(prismaticVerticalJoint, -0.6)
30     end
31     if(h0 == 1 or h1 == 1) then
32         stopHorizontalPosition = sim.getJointPosition(
               prismaticHorizontalJoint)
33         sim.setJointTargetPosition(prismaticHorizontalJoint,
               stopHorizontalPosition)
34     else
35         sim.setJointTargetPosition(prismaticHorizontalJoint, 0.02)
36     end
37 end
38
39 function sysCall_sensing()
40     r0 =sim.checkProximitySensor(rotationStop0, rotationTriggerStop
```

```
        )
41      r1 =sim.checkProximitySensor(rotationStop1, rotationTriggerStop
        )
42      v0 =sim.checkProximitySensor(verticalStop0,
        verticalTriggerBottomStop)
43      v1 =sim.checkProximitySensor(verticalStop1,
        verticalTriggerTopStop)
44      h0 =sim.checkProximitySensor(horizontalStop0,
        horizontalTriggerBackStop)
45      h1 =sim.checkProximitySensor(horizontalStop1,
        horizontalTriggerFrontStop)
46
47      local collisionObjectHandle = sim.getCollisionHandle('
        BaseCollision')
48      local result, pairHandles = sim.handleCollision(
        collisionObjectHandle)
49      if result>0 then
50          print('Robot is colliding. Colliding pair is '..getAsString
            (pairHandles))
51      end
52
53  end
54
55  function sysCall_cleanup()
56  end
```

Thanks to this kind of implementation, the 3D simulations run on CoppeliaSim allow the designers to understand the real robotic arm limits before to act on it. For example, the simulation can be used to understand if a movement can be dangerous for the robot, or it can be used to understand how a movement can be performed (by moving one joint at a time, or more than one).

## 3.6 Inverse Kinematic

The inverse kinematics problem consists of the determination of the joint variables corresponding to a given end-effector position and orientation. The solution to this problem is of fundamental importance in order to transform the motion specifications, assigned to the end-effector in the operational space, into the corresponding joint space motions that allow execution of the desired motion.

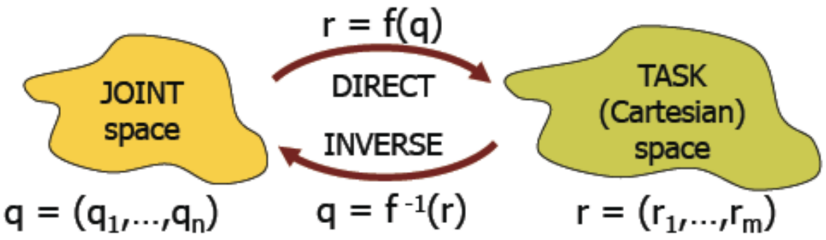



**Figure 3.59:** Inverse Kinematic and Direct Kinematic

Generally the aim is to find a robot configuration $q$ such that $\varphi(q) = y*$. If and only if $\varphi$ is invertible $q* = \varphi^{-1}(y*)$. But in general, $\varphi$ will not be invertible:

- The pre-image $\varphi^{-1}(y*)$ may be empty: no configuration can generate the desired $y*$;
- The pre-image $\varphi^{-1}(y*)$ may be large: many configurations can generate the desired $y*$.

The inverse kinematics problem is very complex for the following reasons:

- it is not always possible to find a closed form solution, in fact the equations to solve are in general nonlinear;
- multiple (depending on the number of degrees of freedom and on the number of non null DH parameters) or infinite (in the case of a kinematically redundant manipulator) solutions may exist;
- there might be no solution, in fact the existence of a solution is guaranteed only if the point belongs to the manipulator dexterous workspace;
- it is not always possible to compute all the admissible solutions (depending on the solution technique applied).

The solutions can be of two kind:

- analytical (in closed form);
- numerical.

The analytical solutions can be:

- geometric inspection;
- algebric methods.

The numerical solutions methods are needed if the manipulator is redundant or if the closed form solution is not available. They use the Jacobian matrix of the direct kinematic map and two different iterative methods can be followed:

- the Newton method;
- the gradient method.

In the Newton method it is important that the Jacobian matrix of the direct kinematic map is not singular, because it has to be inverted in order to compute the inverse kinematic (in the case of robot redundancy the pseudo inverse matrix is used).

The gradient method is based on the iterative computation of the derivative of the error (the objective is to minimize a certain cost function) and on the integration with respect to the initial conditions. It is important to choose a multiplicative factor $\alpha$, called step size, such that the error function decreases at each iteration. It has to be greater than 0, but also not too much great in order to avoid to "miss" the minimum of the function.

### 3.6.1 Inverse Kinematic in CoppeliaSim

Another important usage of CoppeliaSim is for resolving the inverse kinematic. More specifically, given a particular robot model well configured, the simulator is able to make the tip of the robot reach the minimum distance to a specific point in the space (i.e., if the point is inside the workspace of the robot, it is directly reached, so the minimum distance between tip and target point will be zero, else the robot is oriented and moved in order to have the minimum distance between the tip and the target point, but it will be greater than zero).

As already discussed, the inverse kinematic analysis is really useful when it is needed to perform a particular action, due to the fact that starting by a particular point in the *3D*

*space*, it is possible to retrive the same point (if it is inside the workspace) in the *joint space*. This possibility enable the designers to realize particular scenes in which the robot can perform actions which makes it interact with the surrounding environment (i.e., ambience and objects).

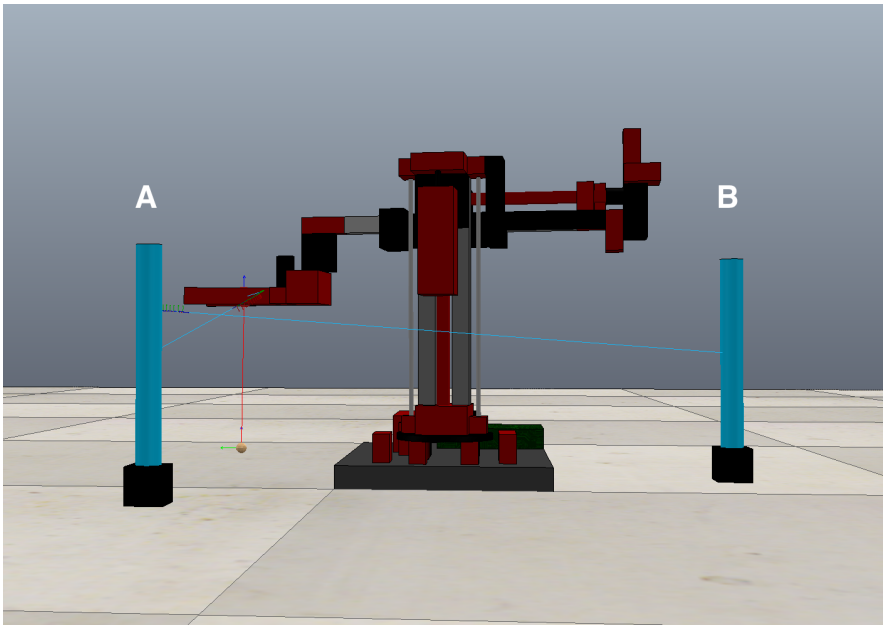



**Figure 3.60:** Demo starting position in scene for Inverse Kinematic

In the case of this project, the presented demo consists of moving an object between two different positions. More specifically, the initial scene of the demo is depicted in Figure 3.60. Here the robot is able to move from its reset position to the one which allow it to pick up a cylinder in position **A** and move it to position **B**. In the end the robot returns in its reset position. The demo is also able to repeat these movements indefinitely up to when a user stops it.

To set the demo on CoppeliaSim, a path was created, in which the five points are located where the tip of the robot should be for performing a specific action (i.e., prepare to pick the cylinder, pick it, move it, release it). Two *dummy* elements are created, one for the tip, which are moved inside the center of the gripper, and one for the target, which is the point the robot needs to follow.

In order to move from direct kinematic to inverse kinematic, it is needed[4] to change the mode for all the joint in the robot, from *Torque/force mode* to *Inverse kinematics mode*.

To enable the target to follow the path[6], a threaded child script, is attached to the target

dummy element. Its code can be read in the Listing 3.8 and uses the function “followPath" to make the target point follow the five-points path already described.

**Listing 3.8:** Threaded child script for follow the path

```
function sysCall_threadmain()
    targetHandle = sim.getObjectHandle('Target')
    pathHandle = sim.getObjectHandle('PathPickRelease')
    sim.followPath(targetHandle, pathHandle, 1, 0, 0.01, 0)
end

function sysCall_cleanup()
end
```

By doing so, it is possible, by only running the simulation, to see if the robot will collide with any object during the execution of the demo and to adjust the trajectory of the points. More important, thanks to the inverse kinematics resolution from CoppeliaSim, it is also possible to retrieve the values for the *joint space* and to use them to manage the real robot to perform the same demo in the real world.

To simplify these operation of reading the joint variables from the simulator, a *non threaded child script* is realized and attached to the first element of the robot which prints the point position in the joint space on the debug panel, as shown in Figure 3.61. It is listed in Listing A.1 and can be analyzed for more information.

It is relevant to notice that the speed of the robot in the simulation is not the same of the one reached by the real robot during movements, for this reason it is not correct to make assumptions on simulation time.

It is important to underline that CoppeliaSim inverse kinematic calculation results are used to manage the real world robot and for this reason, it is important to realize a 3D model and scene with a real good estimate. It is this good approximation which enables the designers to use the data retrieved by the simulator, to manage real world operations.

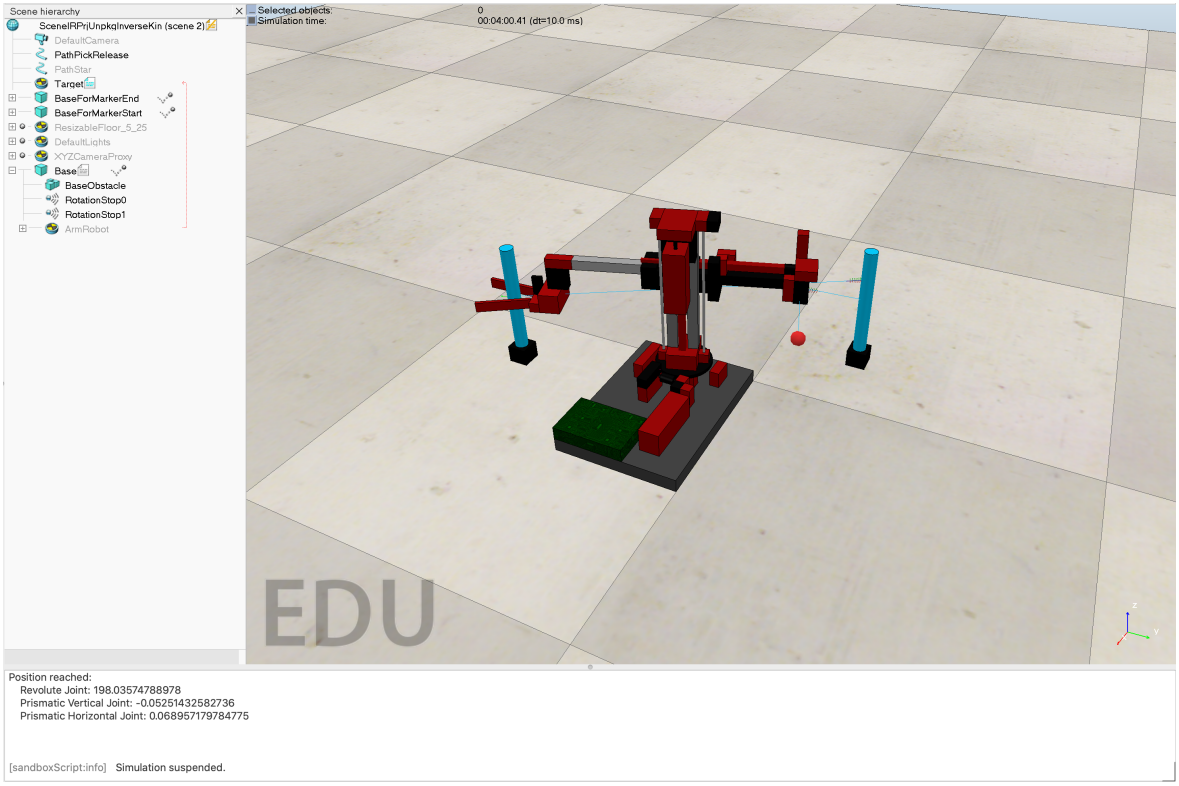



**Figure 3.61:** Simulation of the demo and debug of the point position in the joint space

## 3.7 Test

A simulation scene was realized in CoppeliaSim. Two cylinders were located in two points (A and B). A path was defined: from the initial position of the robot to point A and then from point A to point B. The simulation was executed in order to verify that there were no collision along the path. After the simulation was executed using CoppeliaSim, a Matlab script that makes the robot behave the same way was realized. The script was executed and as expected the robot behaves in the real world the same way of the simulation. At this point the script was updated in order to control the closure and the opening of the gripper, because it was not possible to simulate these actions. A cylinder was placed in point A in real world and the script was executed again. As result the target object was correctly moved from point A to point B. At the end the script was updated again in order to pick up the object from point B and move it to point A. The two operations were closed inside an infinite loop and the script was executed. As result the robot behaves exactly as expected, thus the approximation obtained with the 3D model defined in CoppeliaSim can be considered good. The final version of the script follows:

**Listing 3.9:** main_robot.m

```
1  while true
2      points =[42.19,0,0,gripper.open; 42.19,0.081,-0.06,gripper.open
           ; 42.19,0.081,-0.06,gripper.close; 42.19,0.081,-0.018,
           gripper.close; 198.03,0.069,-0.018,gripper.close;
           198.03,0.069,-0.06,gripper.close; 198.03,0.069,-0.06,gripper
           .open; 198.03,0,-0.06,gripper.open; -5,0,0,gripper.open];
3      for i = 1:size(points,1)
4          old_position = move_robot(t, old_position, 'rotation',
               points(i,1),'horizontal',points(i,2),'vertical',points(i
               ,3),'gripper',points(i,4));
5      end
6      points =[198.03,0,-0.06,gripper.open; 198.03,0.069,-0.06,
           gripper.open; 198.03,0.069,-0.06,gripper.close;
           198.03,0.069,0,gripper.close; 42.19,0.081,0,gripper.close;
           42.19,0.081,-0.06,gripper.close; 42.19,0.081,-0.06,gripper.
           open; 42.19,0,-0.06,gripper.open; -5,0,0,gripper.open];
7      for i = 1:size(points,1)
8          old_position = move_robot(t, old_position, 'rotation',
               points(i,1),'horizontal',points(i,2),'vertical',points(i
               ,3),'gripper',points(i,4));
9      end
10 end
```

# CHAPTER 4

# Encountered Issues

## 4.1 Impossibility to read value from the encoders

### 4.1.1 Issue

Sometime it can happen that, even if the HSC are correctly set to read values from the encoders, they do not work.

### 4.1.2 Solution

This issue can be easily solved by deassembling and removing the encoder box from its slot and rotating by hand the pin of the motor. After that the encoder box can be assembled and put again in its slot.

## 4.2 Sending array of values from Matlab to TIA Portal

### 4.2.1 Issue

A problem that had to be solved during the environment setup for this project was related to the communication between Matlab and TIA Portal. When it was tried to send more than one byte from Matlab to TIA Portal, the values sent could not be read.

### 4.2.2 Solution

The problem was solved [2] by disabling the option "optimized access" of the Data Block in TIA Portal. This option can be disabled by right-clicking on the Data Block object, selecting "properties", then selecting "attributes" in the "general" window and finally unchecking the correspondent box.

## 4.3 Correctly formatting data to be sent from Matlab to TIA Portal

### 4.3.1 Issue

Another encountered problem regarding the communication between Matlab and TIA Portal was to find the correct way to format data in Matlab in order to correctly read them in TIA Portal. In fact it happened that when it was firstly tried to send data, they were not properly read by TIA Portal.

### 4.3.2 Solution

This issue was solved by sending from TIA Portal to Matlab the same data that were expected to be received by TIA Portal. Then the values received by Matlab were read and sent back to TIA Portal, where they were finally correctly read. From this experience it was found that the problem can be related to two different factors:

- an uncorrect offset between sequential data;
- a different representation of the same data type in the two softwares (for example little endian vs big endian).

The offset problem can be adjusted by putting the correct number of zeros between successive data, while the representation problem can be solved by trials and errors, trying to decode the values using different possible common representations until the correct one is found.

## 4.4 Connection Timeout in communication between Matlab and TIA Portal

### 4.4.1 Issue

It can happens that, when one of the two softwares involved in the communication is waiting for some data coming from the other software, the connection timeout expires and the data are not received in time.

### 4.4.2 Solution

This problem can be easily solved by modifying the connection timeout property of the connection object created by Matlab. This operation can be easily done by simply assigning a greater value in seconds. Foe example, assuming that the connection object variable is named "t" and that the wanted timeout is 180 seconds, it can be set with the following line of code:

**Listing 4.1:** Set connection timeout

```
1 t.timeout = 180;
```

## 4.5 Some variables do not store the correct value after system merker activation in TIA Portal

### 4.5.1 Issue

It can happen that if some variables are stored in memory addresses like $MDx$, after the activation of system merkers the values in those variables are not the correct ones.

### 4.5.2 Solution

This problem can be solved easily by assigning to system merkers memory addresses like $My$, respecting the condition $y \neq x$, because the addresses indicated with $Mx$ or $MDx$ point to the same memory location, but they refers to different size: $Mx$ indicates the starting point $x$ with a one-byte size, while $MDx$ indicates the starting point $x$ with an entire-word size.

To set all the memory addresses that are needed for our purpose 16 bits are enough since the greatest value that we can reach is around 4000 for the rotational counter, that is reached

when the counterclockwise rotation stops. Since with 16 bits it is possible to reach values up to 65535, the operational range is covered. For this purpose, in those variables definition is enough to use a MW type (Memory Word) that is a 16 bits variable. The MB variables (Memory Byte) or M variables (Memory bit), related to the same kind of operation, are then assigned in the corresponding memory area. A list of all the data type is reported in the Figure4.1.

| Area operandi | Descrizione | Accesso tramite unità delle seguenti dimensioni | Notazione S7 |
|---|---|---|---|
| **Immagine di processo delle uscite** | La CPU scrive i valori dell'immagine di processo delle uscite nelle unità di uscita. | Uscita (bit) | Q |
| | | Byte di uscita | QB |
| | | Parola di uscita | QW |
| | | Doppia parola di uscita | QD |
| **Immagine di processo degli ingressi** | La CPU legge gli ingressi dalle unità di ingresso e memorizza i valori nell'immagine di processo degli ingressi. | Ingresso (bit) | I |
| | | Byte di ingresso | IB |
| | | Parola di ingresso | IW |
| | | Doppia parola di ingresso | ID |
| **Merker** | Spazio di memoria per i risultati intermedi calcolati nel programma. | Merker (bit) | M |
| | | Byte di merker | MB |
| | | Parola di merker | MW |
| | | Doppia parola di merker | MD |
| **Blocco dati** | Permettono di creare:<br>**DB globali**: area di memoria **statica** condivisa tra più programmi e che permette strutture complesse,<br>**DB di istanza**: area di memoria **statica** assegnata all'istanza di un determinato Function Block. | Bit di dati | DBX |
| | | Byte di dati | DBB |
| | | Parola di dati | DBW |
| | | Doppia parola di dati | DBD |
| **Dati locali** | Area Locale ad un Blocco Programma per la durata della sua elaborazione. | Bit di dati locali | L |
| | | Byte di dati locali | LB |
| | | Parola di dati locali | LW |
| | | Doppia parola di dati locali | LD |

**Figure 4.1:** Memory variable types in Siemens S7

In particular, in this project the memory addresses used are:

# CHAPTER 5

# Conclusion

## 5.1 Reached goals

The goals of the project were reached. In particular:

1. An high level control system of the robotic arm was realized, in fact some high level functions were realized in Matlab in order to communicate with the PLC. The PLC was programmed using TIA Portal in order to connect to Matlab through TCP protocol and to move the robot depending on the values received by Matlab. In addition to that, some other control actions were realized using TIA portal:
   - the emergency stop and the reset of the robot;
   - the control of some LEDs in order to understand the actual state of the robot.
2. A 3D model of the robot was realized in CoppeliaSim and the direct and inverse kinematics were computed. In addition to this, the Matlab high level function was adapted doing some conversions of the input values. The function accepts them in the same form obtained computing the inverse kinematic of the robot in CoppeliaSim and send them to the PLC in the form corresponding to the one obtained by the robot sensors for those values.

3. The model was validated performing a task in the real world. In particular the task was firstly simulated in CoppeliaSim and then was executed in the real world using a Matlab script that executes the corresponding high level function that were defined. As result the behaviour of the robot was the expected one.

## 5.2 Future works

This work can be extended. Some ideas for future works are:

- to update the 3D model in order to have the possibility to use the gripper also in simulations;
- to allow CoppeliaSim to communicate with Matlab in order to automatize the execution of the simulated scenes into the real world;
- to create other simulation scenes and tasks that can be simulated and then executed by the robot.

# APPENDIX A

## Appendix

## A.1 Lua Codes

**Listing A.1:** Non Threaded child script to check distance

```
1  function sysCall_init()
2      revoluteJoint = sim.getObjectHandle('RevoluteJoint')
3      prismaticVerticalJoint = sim.getObjectHandle('
           PrismaticVerticalJoint')
4      prismaticHorizontalJoint = sim.getObjectHandle('
           PrismaticHorizontalJoint')
5      collisionOccurred = false
6      actualDistance = 100
7      justDone = 0
8  end
9
10 function sysCall_actuation()
11 end
12
13 function sysCall_sensing()
```

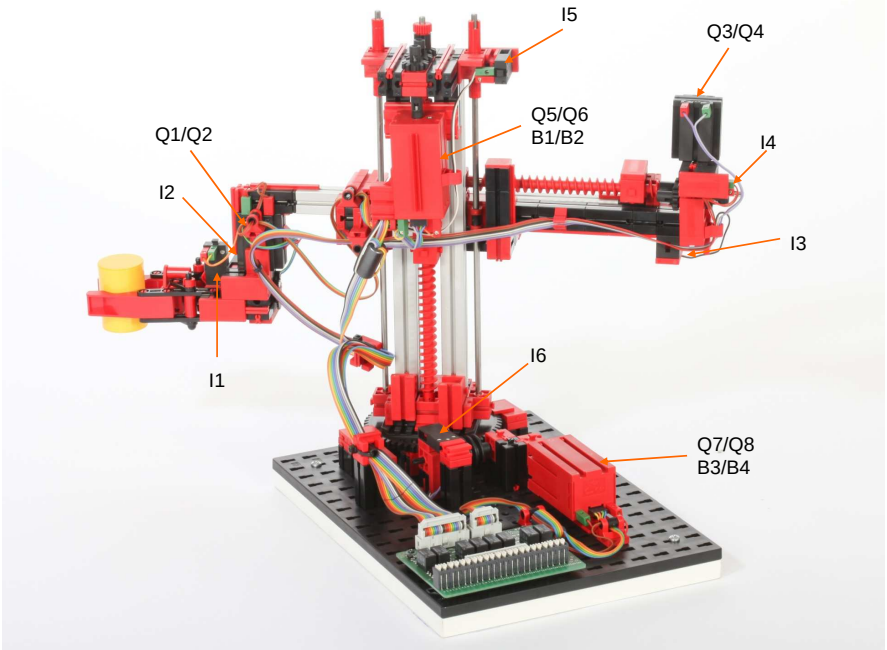
```
14      local collisionObjectHandle = sim.getCollisionHandle('
            BaseCollision')
15      local resultCollision, pairHandles = sim.handleCollision(
            collisionObjectHandle)
16      if resultCollision>0 then
17          collisionOccurred = true
18      end
19
20      local tip = sim.getObjectHandle('Tip')
21      local target = sim.getObjectHandle('Target')
22
23      local resultDistance,distData,objectPair=sim.checkDistance(tip,
            target,0)
24      if resultDistance>0 and actualDistance ~= distData[7] then
25          actualDistance = distData[7]
26          justDone = 0
27      elseif actualDistance == distData[7] and justDone < 10 then
28          justDone = justDone + 1
29      elseif justDone == 10 then
30          if(collisionOccurred) then
31              print('A collision occurred')
32              collisionOccurred = false
33          end
34          if(distData[7] < 0.0016) then
35              print('Position reached: ')
36          else
37              print('Position cannot be reached: ')
38          end
39          print('\tRevolute Joint: '.. sim.getJointPosition(
                revoluteJoint)/math.pi*180)
40          print('\tPrismatic Vertical Joint: '.. sim.getJointPosition
```

```
            (prismaticVerticalJoint))
41          print('\tPrismatic Horizontal Joint: '.. sim.
                getJointPosition(prismaticHorizontalJoint))
42          print('\n\n')
43          justDone = justDone + 1
44      end
45  end
46
47  function sysCall_cleanup()
48  end
```

## A.2 Datasheets

**511938 3D-Robot TX 24V**

fischertechnik

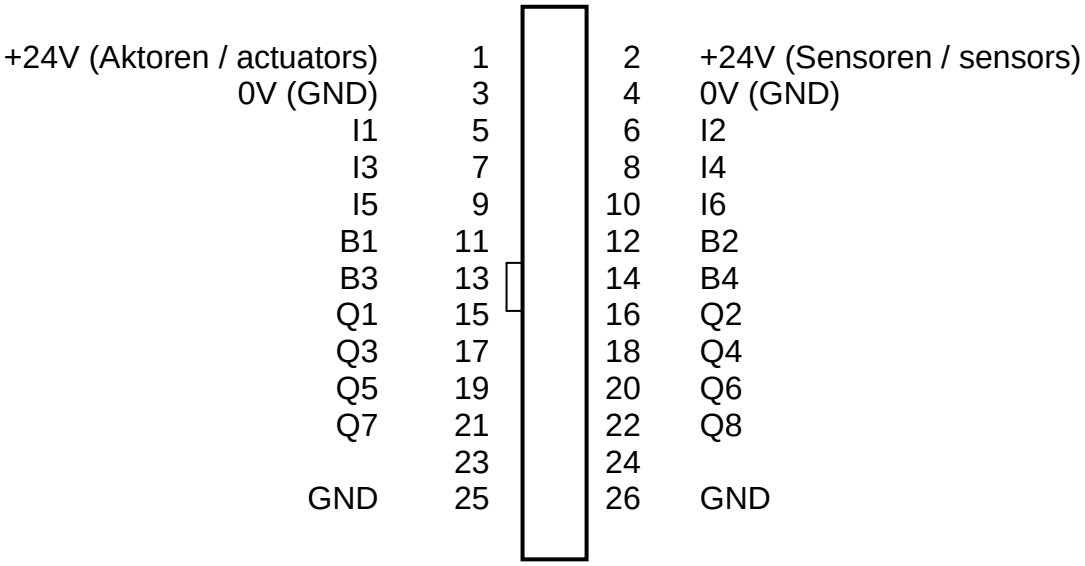


Belegungsplan für 3-D-Robot 24V (Art.-Nr. 511938)
Circuit layout for 3-D-Robot 24V (item-no. 511938)

| **Klemme Nr.**<br>**Terminal no.** | **Funktion**<br>**Function** | **Eingang/Ausgang**<br>**Input/Output** |
|---|---|---|
| 1 | Stromversorgung (+) Aktoren<br>power supply (+) actuators | 24V DC |
| 2 | Stromversorgung (+) Sensoren<br>power supply (+) sensors | 24V DC |
| 3 | Stromversorgung (-)<br>power supply (-) | 0V |
| 4 | Stromversorgung (-)<br>power supply (-) | 0V |
| 5 | Referenztaster Greifer<br>reference switch claw | I1 |
| 6 | Impulstaster Greifer<br>pulse counter gripper | I2 |
| 7 | Referenztaster Greifarm<br>reference switch grip arm | I3 |
| 8 | Impulstaster Greifarm<br>pulse counter grip arm | I4 |
| 9 | Referenztaster Vertikalachse<br>reference switch vertical axis | I5 |
| 10 | Referenztaster Drehkranz<br>reference switch turntable | I6 |
| 11 | Encoder Vertikalachse Impuls 1<br>encoder vertical axis impuls 1 | B1 |
| 12 | Encoder Vertikalachse Impuls 2<br>encoder vertical axis impuls 2 | B2 |
| 13 | Encoder Drehkranz Impuls 1<br>encoder turntable impuls 1 | B3 |
| 14 | Encoder Drehkranz Impuls 2<br>encoder turntable impuls 2 | B4 |
| 15 | Motor Greifer öffnen<br>motor gripper open | Q1 (M1) |
| 16 | Motor Greifer schließen<br>motor gripper close | Q2 (M1) |
| 17 | Motor Greifarm vor<br>motor grip arm befor | Q3 (M2) |

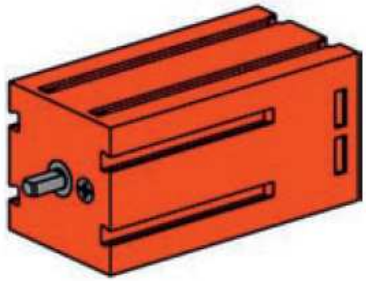
| | | |
|---|---|---|
| 18 | Motor Greifarm zurück<br>motor grip arm back | Q4 (M2) |
| 19 | Motor Vertikalachse abwärts<br>motor vertical axis down | Q5 (M3) |
| 20 | Motor Vertikalachse aufwärts<br>motor vertical axis up | Q6 (M3) |
| 21 | Motor Drehkranz rechts<br>motor turntable right | Q7 (M4) |
| 22 | Motor Drehkranz links<br>motor turntable left | Q8 (M4) |

**26pol. Steckerleiste**

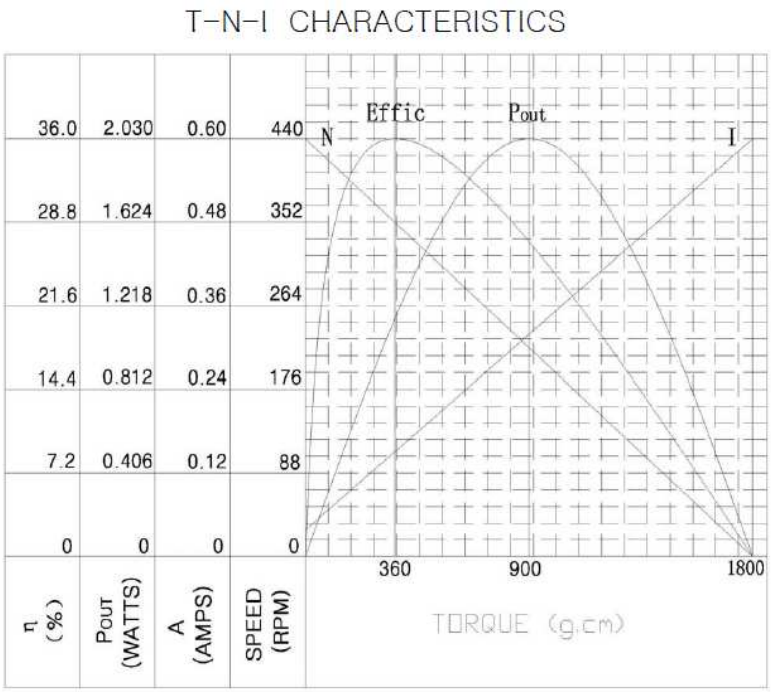


FT-T-KN
7.11.2011

# Datenblatt Encodermotor 24V Art.-Nr. 144643
# *Datasheet Encoder motor 24V Art. No. 144643*

| | |
|---|---|
|  | Maße/*dimensions*: 60x30x30mm<br><br>Abtriebswelle/*output shaft*: D=4mm, L=7,5mm,<br>2 Abflachungen je 0,7mm/ *2 bevels 0.7mm each*<br><br>Stromversorgung: 24VDC über 2 fischertechnik Anschlussbuchsen D=2,5mm/<br>*Power supply: 24VDC with 2 fischertechnik connection sockets D=2.5mm* |

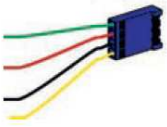
**Motordaten/*motor data:***



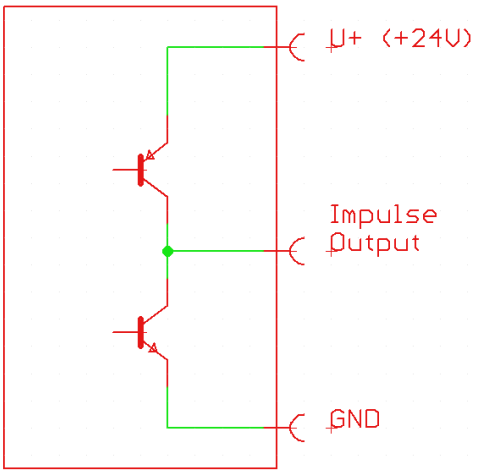

**Encoderdaten/*encoder data:***

Quadraturencoder,
Stromversorung/power supply 24VDC
Signal: Quadratur Encoder, Push-Pull Output (0/24V), max. 10mA
Frequenz/frequency: max 1kHz.

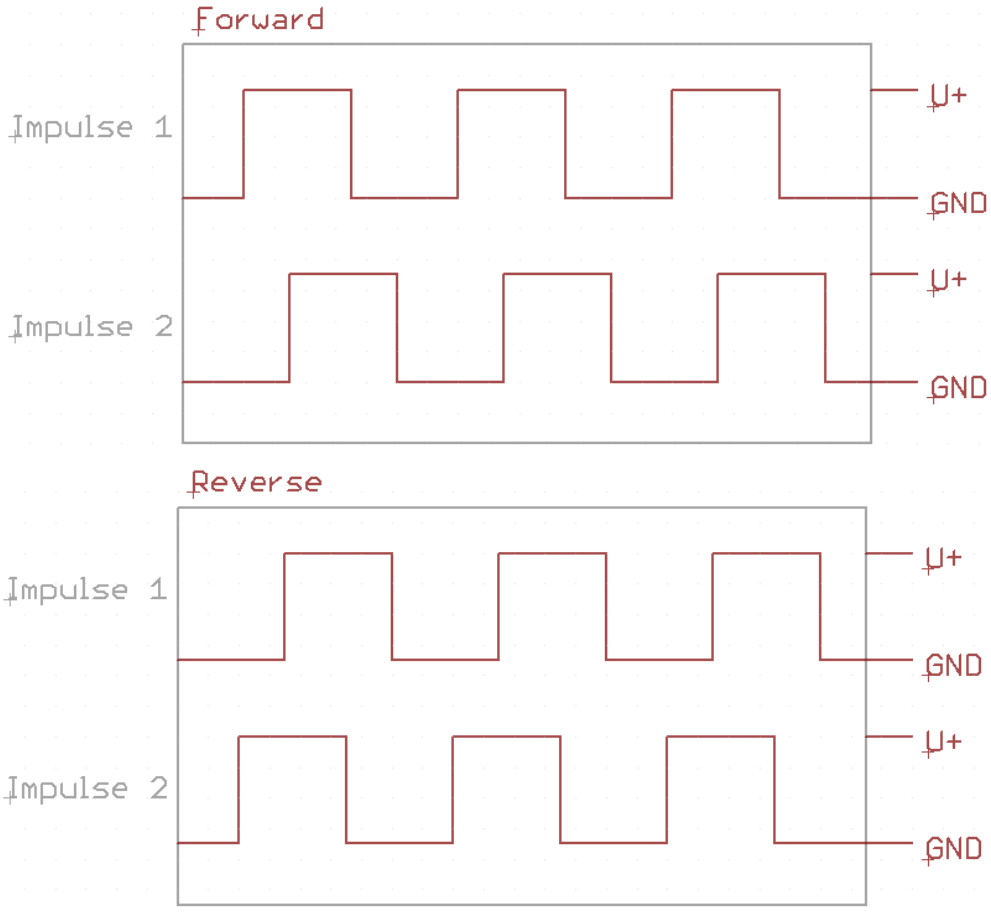
Anschlüsse/*connector*: 4-pol Stiftleiste,
passendes Kabel/*fitting cable*: Art.No. 119785
rot/*red*=+24V, grün/*green*=0V, schwarz/*black*=Puls1,
gelb/*yellow*=Impuls2



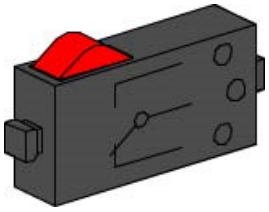
Ersatzschaltbild eines 24V Encoder-Ausgangs (2 pro Motor)
*Equivalent circuit diagram of 24V encoder output (2 per motor)*



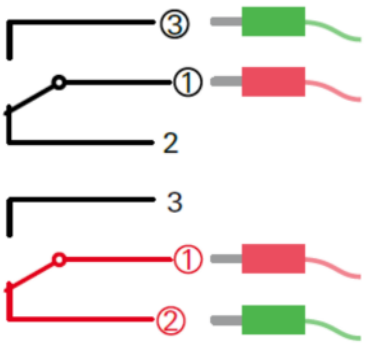

Ausgangssignal des 24V-Encoders für verschiedene Drehrichtungen.
*Output signal of 24V-encoder for different directions of rotation*



FT-T-KN
2017-08-03

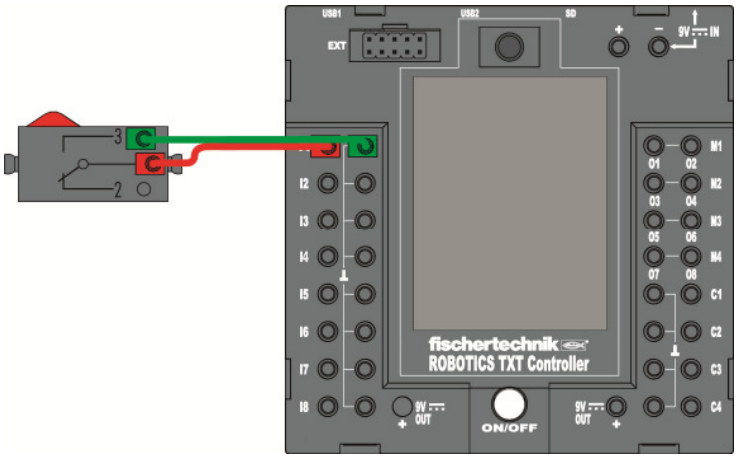
| | | |
|---|---|---|
| **Art.-Nr.**<br>***part. no.*** | **37783** |  |
| **Bezeichnung**<br>***name*** | **Mini-Taster**<br>***Mini switch*** | |
| Abmessungen<br>*dimensions:* | 30x15x7,5mm | |
| Gewicht<br>*weight:* | 3,4g | |
| Schaltleistung:<br>*Switching power:* | max. 2A, 50V | |

Signal: Digitaler Schalter 0 / 1, als Öffner oder Schließer verwendbar
*Signal: Digital switch 0 /1, can be used aus "normally open" or "normally closed"*




Anschlüsse: Kontakte 1 und 3: "Schließer", Kontakte 1 und 2 angeschlossen: "Öffner"
*Connections: Contacts 1 and 3: NO (normally open) Contacts 1 and 2: NC (normally closed)*

Anschluss an TXT Controller an Eingängen I1-I8, Eingangsart: Digital 5kΩ
*Connection to TXT Controller at Inputs I1-I8, Input mode: digital 5kΩ*



# 6ES7215-1BG40-0XB0

## CPU 1215C, AC/DC/RLY, 14DI/10DO/2AI/2AO

Product details | Technical data | CAx data

### Technical data

(https://www.automation.siemens.com/bilddb/index.aspx?objKey=G_ST70_XX_00916)

SIMATIC S7-1200, CPU 1215C, compact CPU, AC/DC/relay, 2 PROFINET ports, onboard I/O: 14 DI 24 V DC; 10 DO relay 2 AI 0-10 V DC, 2 AO 0-20 mA DC, Power supply: AC 85-264 V AC at 47-63 Hz, Program/data memory 125 KB

| | |
|---|---|
| **General information** | |
| Product type designation | CPU 1215C AC/DC/relay |
| Engineering with | |
| ● Programming package | STEP 7 V17 or higher |
| **Supply voltage** | |
| Rated value (AC) | |
| ● 120 V AC | Yes |
| ● 230 V AC | Yes |
| permissible range, lower limit (AC) | 85 V |
| permissible range, upper limit (AC) | 265 V |
| Line frequency | |
| ● permissible range, lower limit | 47 Hz |
| ● permissible range, upper limit | 63 Hz |
| **Input current** | |
| Current consumption (rated value) | 100 mA at 120 V AC; 50 mA at 240 V AC |
| Current consumption, max. | 300 mA at 120 V AC; 150 mA at 240 V AC |
| Inrush current, max. | 20 A; at 264 V |
| $I^2t$ | 0.8 $A^2 \cdot s$ |
| **Output current** | |
| for backplane bus (5 V DC), max. | 1 600 mA; Max. 5 V DC for SM and CM |
| **Encoder supply** | |
| 24 V encoder supply | |
| ● 24 V | 20.4 to 28.8V |
| **Power loss** | |
| Power loss, typ. | 14 W |
| **Memory** | |
| Work memory | |
| ● integrated | 125 kbyte |
| ● expandable | No |
| Load memory | |
| ● integrated | 4 Mbyte |
| ● Plug-in (SIMATIC Memory Card), max. | with SIMATIC memory card |
| Backup | |
| ● present | Yes |
| ● maintenance-free | Yes |
| ● without battery | Yes |
| **CPU processing times** | |
| for bit operations, typ. | 0.08 µs; / instruction |
| for word operations, typ. | 1.7 µs; / instruction |
| for floating point arithmetic, typ. | 2.3 µs; / instruction |
| **CPU-blocks** | |
| Number of blocks (total) | DBs, FCs, FBs, counters and timers. The maximum number of addressable blocks ranges from 1 to 65535. There is no restriction, the entire working memory can be used |
| OB | |
| ● Number, max. | Limited only by RAM for code |
| **Data areas and their retentivity** | |
| Retentive data area (incl. timers, counters, flags), max. | 14 kbyte |
| Flag | |
| ● Size, max. | 8 kbyte; Size of bit memory address area |
| Local data | |
| ● per priority class, max. | 16 kbyte; Priority class 1 (program cycle): 16 KB, priority class 2 to 26: 6 KB |
| **Address area** | |
| Process image | |
| ● Inputs, adjustable | 1 kbyte |
| ● Outputs, adjustable | 1 kbyte |
| **Hardware configuration** | |
| Number of modules per system, max. | 3 comm. modules, 1 signal board, 8 signal modules |
| **Time of day** | |

| | |
|---|---|
| Clock | 0.0.0.0 |
| ● Hardware clock (real-time) | Yes |
| ● Backup time | 480 h; Typical |
| ● Deviation per day, max. | ±60 s/month at 25 °C |
| **Digital inputs** | |
| Number of digital inputs | 14; Integrated |
| ● of which inputs usable for technological functions | 6; HSC (High Speed Counting) |
| Source/sink input | Yes |
| Number of simultaneously controllable inputs | |
| all mounting positions | |
| — up to 40 °C, max. | 14 |
| Input voltage | |
| ● Rated value (DC) | 24 V |
| ● for signal "0" | 5 V DC at 1 mA |
| ● for signal "1" | 15 V DC at 2.5 mA |
| Input delay (for rated value of input voltage) | |
| for standard inputs | |
| — parameterizable | Yes; 0.2 ms, 0.4 ms, 0.8 ms, 1.6 ms, 3.2 ms, 6.4 ms and 12.8 ms, selectable in groups of four |
| — at "0" to "1", min. | 0.2 ms |
| — at "0" to "1", max. | 12.8 ms |
| for interrupt inputs | |
| — parameterizable | Yes |
| for technological functions | |
| — parameterizable | Single phase: 3 @ 100 kHz & 3 @ 30 kHz, differential: 3 @ 80 kHz & 3 @ 30 kHz |
| Cable length | |
| ● shielded, max. | 500 m; 50 m for technological functions |
| ● unshielded, max. | 300 m; for technological functions: No |
| **Digital outputs** | |
| Number of digital outputs | 10; Relays |
| Switching capacity of the outputs | |
| ● with resistive load, max. | 2 A |
| ● on lamp load, max. | 30 W with DC, 200 W with AC |
| Output delay with resistive load | |
| ● "0" to "1", max. | 10 ms; max. |
| ● "1" to "0", max. | 10 ms; max. |
| Relay outputs | |
| ● Number of relay outputs | 10 |
| ● Number of operating cycles, max. | mechanically 10 million, at rated load voltage 100 000 |
| Cable length | |
| ● shielded, max. | 500 m |
| ● unshielded, max. | 150 m |
| **Analog inputs** | |
| Number of analog inputs | 2 |
| Input ranges | |
| ● Voltage | Yes |
| Input ranges (rated values), voltages | |
| ● 0 to +10 V | Yes |
| — Input resistance (0 to 10 V) | ≥100k ohms |
| Cable length | |
| ● shielded, max. | 100 m; twisted and shielded |
| **Analog outputs** | |
| Number of analog outputs | 2 |
| Output ranges, current | |
| ● 0 to 20 mA | Yes |
| **Analog value generation for the inputs** | |
| Integration and conversion time/resolution per channel | |
| ● Resolution with overrange (bit including sign), max. | 10 bit |
| ● Integration time, parameterizable | Yes |
| ● Conversion time (per channel) | 625 µs |
| **Encoder** | |
| Connectable encoders | |
| ● 2-wire sensor | Yes |
| **1. Interface** | |
| Isolated | Yes |
| automatic detection of transmission rate | Yes |
| Autonegotiation | Yes |
| Autocrossing | Yes |
| Interface types | |
| ● RJ 45 (Ethernet) | Yes |
| ● Number of ports | 2 |
| ● integrated switch | Yes |
| Protocols | |
| ● PROFINET IO Controller | Yes |
| ● PROFINET IO Device | Yes |
| ● SIMATIC communication | Yes |
| ● Open IE communication | Yes; Optionally also encrypted |
| ● Web server | Yes |
| ● Media redundancy | Yes |
| PROFINET IO Controller | |
| ● Transmission rate, max. | 100 Mbit/s |
| Services | |
| — PG/OP communication | Yes; encryption with TLS V1.3 pre-selected |
| — Isochronous mode | No |
| — IRT | No |
| — PROFIenergy | No |
| — Prioritized startup | Yes |
| — Number of IO devices with prioritized startup, max. | 16 |
| — Number of connectable IO Devices, max. | 16 |
| — Number of connectable IO Devices for RT, max. | 16 |

| | |
|---|---|
| — of which in line, max. | 16 |
| — Activation/deactivation of IO Devices | Yes |
| — Number of IO Devices that can be simultaneously activated/deactivated, max. | 8 |
| — Updating time | The minimum value of the update time also depends on the communication component set for PROFINET IO, on the numb IO devices and the quantity of configured user data. |
| PROFINET IO Device | |
| Services | |
| — PG/OP communication | Yes; encryption with TLS V1.3 pre-selected |
| — Isochronous mode | No |
| — IRT | No |
| — PROFIenergy | Yes |
| — Shared device | Yes |
| — Number of IO Controllers with shared device, max. | 2 |
| **Protocols** | |
| Supports protocol for PROFINET IO | Yes |
| PROFIBUS | Yes; CM 1243-5 (master) or CM 1242-5 (slave) required |
| OPC UA | Yes; OPC UA Server |
| AS-Interface | Yes; CM 1243-2 required |
| Protocols (Ethernet) | |
| ● TCP/IP | Yes |
| ● DHCP | No |
| ● SNMP | Yes |
| ● DCP | Yes |
| ● LLDP | Yes |
| Redundancy mode | |
| Media redundancy | |
| — MRP | Yes; as MRP redundancy manager and/or MRP client |
| Open IE communication | |
| ● TCP/IP | Yes |
| — Data length, max. | 8 kbyte |
| ● ISO-on-TCP (RFC1006) | Yes |
| — Data length, max. | 8 kbyte |
| ● UDP | Yes |
| — Data length, max. | 1 472 byte |
| Web server | |
| ● supported | Yes |
| ● User-defined websites | Yes |
| OPC UA | |
| ● Runtime license required | Yes; "Basic" license required |
| ● OPC UA Server | Yes; data access (read, write, subscribe), method call, runtime license required |
| — Application authentication | Available security policies: None, Basic128Rsa15, Basic256Rsa15, Basic256Sha256 |
| — User authentication | "anonymous" or by user name & password |
| — Number of sessions, max. | 10 |
| — Number of subscriptions per session, max. | 50 |
| — Sampling interval, min. | 100 ms |
| — Publishing interval, min. | 200 ms |
| — Number of server methods, max. | 20 |
| — Number of monitored items, max. | 1 000 |
| — Number of server interfaces, max. | 2 |
| — Number of nodes for user-defined server interfaces, max. | 2 000 |
| Further protocols | |
| ● MODBUS | Yes |
| **Communication functions** | |
| S7 communication | |
| ● supported | Yes |
| ● as server | Yes |
| ● as client | Yes |
| ● User data per job, max. | See online help (S7 communication, user data size) |
| Number of connections | |
| ● overall | PG Connections: 4 reserved / 4 max; HMI Connections: 12 reserved / 18 max; S7 Connections: 8 reserved / 14 max; Open User Connections: 8 reserved / 14 max; Web Connections: 2 reserved / 30 max; OPC UA Connections: 0 reserved / 10 ma Total Connections: 34 reserved / 64 max |
| **Test commissioning functions** | |
| Status/control | |
| ● Status/control variable | Yes |
| ● Variables | Inputs/outputs, memory bits, DBs, distributed I/Os, timers, counters |
| Forcing | |
| ● Forcing | Yes |
| Diagnostic buffer | |
| ● present | Yes |
| Traces | |
| ● Number of configurable Traces | 2 |
| ● Memory size per trace, max. | 512 kbyte |
| **Interrupts/diagnostics/status information** | |
| Diagnostics indication LED | |
| ● RUN/STOP LED | Yes |
| ● ERROR LED | Yes |
| ● MAINT LED | Yes |
| **Integrated Functions** | |
| Frequency measurement | Yes |
| controlled positioning | Yes |
| Number of position-controlled positioning axes, max. | 8 |
| Number of positioning axes via pulse-direction interface | Up to 4 with SB 1222 |
| PID controller | Yes |
| Number of alarm inputs | 4 |
| **Potential separation** | |
| Potential separation digital inputs | |
| ● Potential separation digital inputs | 500V AC for 1 minute |
| ● between the channels, in groups of | 1 |
| Potential separation digital outputs | |

| | |
|---|---|
| ● Potential separation digital outputs | Relays |
| ● between the channels | No |
| ● between the channels, in groups of | 2 |
| **EMC** | |
| Interference immunity against discharge of static electricity | |
| ● Interference immunity against discharge of static electricity acc. to IEC 61000-4-2 | Yes |
| — Test voltage at air discharge | 8 kV |
| — Test voltage at contact discharge | 6 kV |
| Interference immunity to cable-borne interference | |
| ● Interference immunity on supply lines acc. to IEC 61000-4-4 | Yes |
| ● Interference immunity on signal cables acc. to IEC 61000-4-4 | Yes |
| Interference immunity against voltage surge | |
| ● Interference immunity on supply lines acc. to IEC 61000-4-5 | Yes |
| Interference immunity against conducted variable disturbance induced by high-frequency fields | |
| ● Interference immunity against high-frequency radiation acc. to IEC 61000-4-6 | Yes |
| Emission of radio interference acc. to EN 55 011 | |
| ● Limit class A, for use in industrial areas | Yes; Group 1 |
| ● Limit class B, for use in residential areas | Yes; When appropriate measures are used to ensure compliance with the limits for Class B according to EN 55011 |
| **Standards, approvals, certificates** | |
| CE mark | Yes |
| UL approval | Yes |
| cULus | Yes |
| FM approval | Yes |
| RCM (formerly C-TICK) | Yes |
| KC approval | Yes |
| Marine approval | Yes |
| **Ambient conditions** | |
| Free fall | |
| ● Fall height, max. | 0.3 m; five times, in product package |
| Ambient temperature during operation | |
| ● min. | -20 °C |
| ● max. | 60 °C; Number of simultaneously activated inputs or outputs 7 or 5 (no adjacent points) at 60 °C horizontal or 50 °C vertica or 10 at 55 °C horizontal or 45 °C vertical |
| ● horizontal installation, min. | -20 °C |
| ● horizontal installation, max. | 60 °C |
| ● vertical installation, min. | -20 °C |
| ● vertical installation, max. | 50 °C |
| Ambient temperature during storage/transportation | |
| ● min. | -40 °C |
| ● max. | 70 °C |
| Air pressure acc. to IEC 60068-2-13 | |
| ● Operation, min. | 795 hPa |
| ● Operation, max. | 1 080 hPa |
| ● Storage/transport, min. | 660 hPa |
| ● Storage/transport, max. | 1 080 hPa |
| Altitude during operation relating to sea level | |
| ● Installation altitude, min. | -1 000 m |
| ● Installation altitude, max. | 5 000 m; Restrictions for installation altitudes > 2 000 m, see manual |
| Relative humidity | |
| ● Operation, max. | 95 %; no condensation |
| Vibrations | |
| ● Vibration resistance during operation acc. to IEC 60068-2-6 | 2 g (m/s²) wall mounting, 1 g (m/s²) DIN rail |
| ● Operation, tested according to IEC 60068-2-6 | Yes |
| Shock testing | |
| ● tested according to IEC 60068-2-27 | Yes; IEC 68, Part 2-27 half-sine: strength of the shock 15 g (peak value), duration 11 ms |
| Pollutant concentrations | |
| ● SO2 at RH < 60% without condensation | S02: < 0.5 ppm; H2S: < 0.1 ppm; RH < 60% condensation-free |
| **Configuration** | |
| Programming | |
| Programming language | |
| — LAD | Yes |
| — FBD | Yes |
| — SCL | Yes |
| Know-how protection | |
| ● User program protection/password protection | Yes |
| ● Copy protection | Yes |
| ● Block protection | Yes |
| Access protection | |
| ● protection of confidential configuration data | Yes |
| ● Protection level: Write protection | Yes |
| ● Protection level: Read/write protection | Yes |
| ● Protection level: Complete protection | Yes |
| Cycle time monitoring | |
| ● adjustable | Yes |
| **Dimensions** | |
| Width | 130 mm |
| Height | 100 mm |
| Depth | 75 mm |
| **Weights** | |
| Weight, approx. | 550 g |
| **last modified:** | 4/12/2021 |

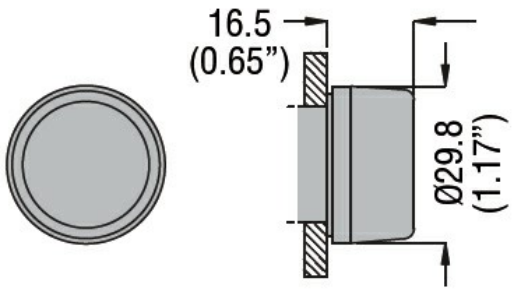
Technical data for the product is also available in the following languages:
› German
› French
› Italian
› Spanish
› Chinese
› Russian

# 8LM2TIL103

PILOT LIGHT HEAD, Ø22MM 8LM METAL SERIES, GREEN

| | | | |
|---|---|---|---|
| Product designation | | | Pilot light heads |
| Product type designation | | | 8LM2TIL10 |
| **General characteristics** | | | |
| Colour | | | Green |
| Material | | | Aluminium and zinc alloy |
| IEC degree of protection | | | IP65 |
| **Electrical characteristics** | | | |
| Max power | | W | 1.2 |
| **Mechanical features** | | | |
| Mounting adapter | | | LM2TAU120 |
| Weight | | g | 24 |
| **UL technical data** | | | |
| UL degree of protection | | | Type 1, 2, 3R, 4, 4X, 12, 12K |
| **Ambient conditions** | | | |
| Temperature | | | |
| | Operating temperature | | |
| | min | °C | -25 |
| | max | °C | +60 |
| | Storage temperature | | |
| | min | °C | -40 |
| | max | °C | +70 |

## Dimensions




## Certifications and compliance

| | |
|---|---|
| Compliance | |
| | CSA C22.2 n° 14 |
| | IEC/EN 60947-1 |
| | IEC/EN 60947-5-1 |
| | UL508 |
| Certificates | |
| | CCC |
| | cULus |
| | EAC |
| | LROS |

| | |
|---|---|
| | RINA |
| | UL |

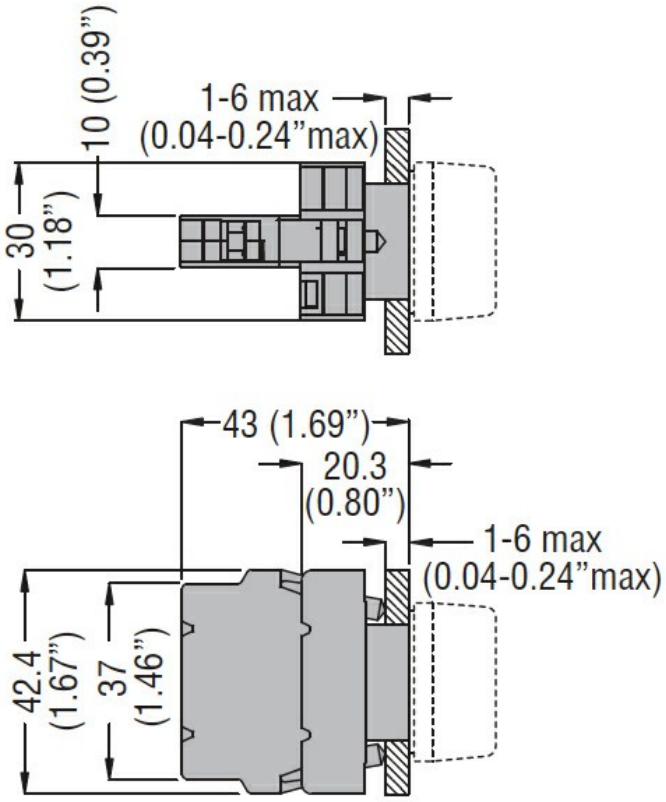
## ETIM classification

| | |
|---|---|
| ETIM 8.0 | EC002024 - Accessories/spare parts for command devices |

# 8LM2TLB3

LED INTEGRATED LAMP-HOLDER, STEADY OR FLASHING LIGHT, Ø22MM 8LM METAL SERIES, STEADY LIGHT, 12...30VAC/DC. GREEN

| | | | | |
|---|---|---|---|---|
| Product designation | | | | LED elements steady light |
| Product type designation | | | | 8LM2TL |
| **General characteristics** | | | | |
| Colour | | | | Green |
| **Electrical characteristics** | | | | |
| Rated frequency | | | Hz | 50/60 |
| Supply voltage (AC) | | | V | 12…30 |
| Supply voltage (DC) | | | V | 12…30 |
| **Operations** | | | | |
| Electrical life | | | hours | 100000 |
| **Mechanical features** | | | | |
| Tightening torque for terminals max | | | Nm | 1 |
| Conductor section | | | | |
| | AWG/kcmil conductor section | | | |
| | | max | | 12 |
| | IEC | | | |
| | | max | mm² | 1 or 2 2.5 |
| Mounting adapter | | | | LM2TAU120 |
| Weight | | | g | 16 |
| **Ambient conditions** | | | | |
| Temperature | | | | |
| | Operating temperature | | | |
| | | min | °C | -25 |
| | | max | °C | +60 |
| | Storage temperature | | | |
| | | min | °C | -40 |
| | | max | °C | +85 |
| **Dimensions** | | | | |




## Wiring diagrams


## Certifications and compliance

| | |
|---|---|
| Compliance | |
| | CSA C22.2 n° 14 |
| | IEC/EN 60947-1 |
| | IEC/EN 60947-5-1 |
| | UL508 |
| Certificates | |
| | CCC |
| | cULus |
| | EAC |

## ETIM classification

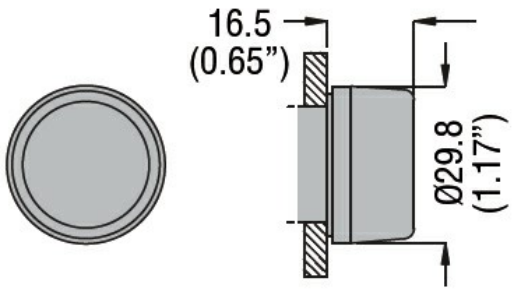
| | |
|---|---|
| ETIM 8.0 | EC000272 - Indicator light complete |

# 8LM2TIL104

PILOT LIGHT HEAD, Ø22MM 8LM METAL SERIES, RED

| | | | |
|---|---|---|---|
| Product designation | | | Pilot light heads |
| Product type designation | | | 8LM2TIL10 |
| **General characteristics** | | | |
| Colour | | | Red |
| Material | | | Aluminium and zinc alloy |
| IEC degree of protection | | | IP65 |
| **Electrical characteristics** | | | |
| Max power | | W | 1.2 |
| **Mechanical features** | | | |
| Mounting adapter | | | LM2TAU120 |
| Weight | | g | 24 |
| **UL technical data** | | | |
| UL degree of protection | | | Type 1, 2, 3R, 4, 4X, 12, 12K |
| **Ambient conditions** | | | |
| Temperature | | | |
| | Operating temperature | | |
| | min | °C | -25 |
| | max | °C | +60 |
| | Storage temperature | | |
| | min | °C | -40 |
| | max | °C | +70 |

## Dimensions




## Certifications and compliance

| | |
|---|---|
| Compliance | |
| | CSA C22.2 n° 14 |
| | IEC/EN 60947-1 |
| | IEC/EN 60947-5-1 |
| | UL508 |
| Certificates | |
| | CCC |
| | cULus |
| | EAC |
| | LROS |

| | |
|---|---|
| | RINA |
| | UL |

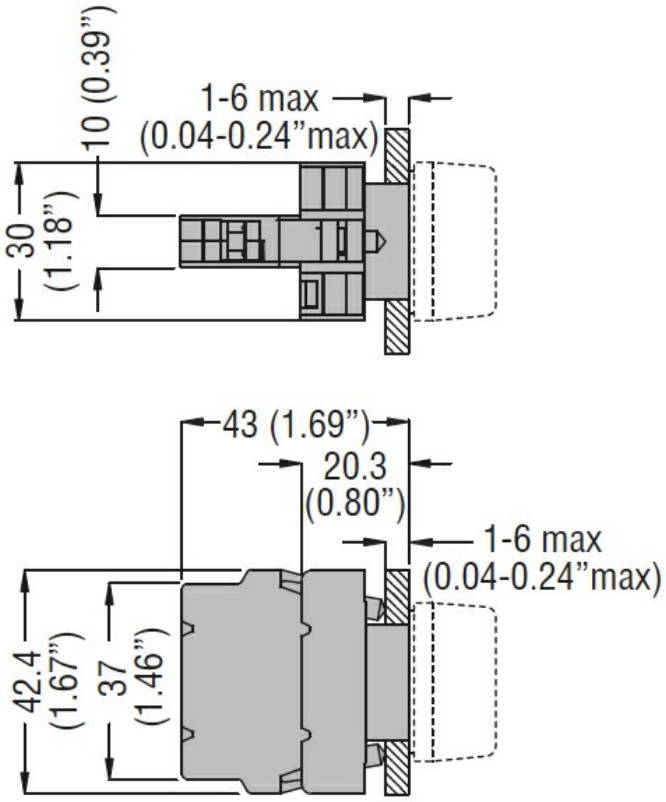
## ETIM classification

| | |
|---|---|
| ETIM 8.0 | EC002024 - Accessories/spare parts for command devices |

# 8LM2TLB4

LED INTEGRATED LAMP-HOLDER, STEADY OR FLASHING LIGHT, Ø22MM 8LM METAL SERIES, STEADY LIGHT, 12...30VAC/DC. RED

| | | | | |
|---|---|---|---|---|
| Product designation | | | | LED elements steady light |
| Product type designation | | | | 8LM2TL |
| **General characteristics** | | | | |
| Colour | | | | Red |
| **Electrical characteristics** | | | | |
| Rated frequency | | | Hz | 50/60 |
| Supply voltage (AC) | | | V | 12…30 |
| Supply voltage (DC) | | | V | 12…30 |
| **Operations** | | | | |
| Electrical life | | | hours | 100000 |
| **Mechanical features** | | | | |
| Tightening torque for terminals max | | | Nm | 1 |
| Conductor section | | | | |
| | AWG/kcmil conductor section | | | |
| | | max | | 12 |
| | IEC | | | |
| | | max | mm² | 1 or 2 2.5 |
| Mounting adapter | | | | LM2TAU120 |
| Weight | | | g | 16 |
| **Ambient conditions** | | | | |
| Temperature | | | | |
| | Operating temperature | | | |
| | | min | °C | -25 |
| | | max | °C | +60 |
| | Storage temperature | | | |
| | | min | °C | -40 |
| | | max | °C | +85 |
| **Dimensions** | | | | |




## Wiring diagrams


## Certifications and compliance

Compliance

| |
|---|
| CSA C22.2 n° 14 |
| IEC/EN 60947-1 |
| IEC/EN 60947-5-1 |
| UL508 |

Certificates

| |
|---|
| CCC |
| cULus |
| EAC |

## ETIM classification

| | |
|---|---|
| ETIM 8.0 | EC000272 - Indicator light complete |

Lovato electric

ENERGY AND AUTOMATION

# LM2T METAL PUSHBUTTONS DATASHEET

| Order code | Colour | | Qty per pkg | Wt |
|---|---|---|---|---|
| | | | n° | [kg] |
| Flush (without mounting adapter). Spring return. | | | | |
| **8 LM2T B102** | Black | | 10 | 0.033 |
| **8 LM2T B103** | Green | | 10 | 0.033 |
| **8 LM2T B104** | Red | | 10 | 0.033 |
| **8 LM2T B105** | Yellow | | 10 | 0.033 |
| **8 LM2T B106** | Blue | | 10 | 0.033 |
| **8 LM2T B108** | White | | 10 | 0.033 |
| Extended (without mounting adapter). Spring return. | | | | |
| **8 LM2T B202** | Black | | 10 | 0.035 |
| **8 LM2T B203** | Green | | 10 | 0.035 |
| **8 LM2T B204** | Red | | 10 | 0.035 |
| **8 LM2T B205** | Yellow | | 10 | 0.035 |
| **8 LM2T B206** | Blue | | 10 | 0.035 |
| **8 LM2T B208** | White | | 10 | 0.035 |

| | | | | |
|---|---|---|---|---|
| **8 LM2T B1102** | O | Black | 10 | 0.033 |
| **8 LM2T B1104** | | Red | 10 | 0.033 |
| **8 LM2T B1113** | I | Green | 10 | 0.033 |
| **8 LM2T B1118** | | White | 10 | 0.033 |

SPRING RETURN.
Ø40mm/1.6" (without mounting adapter).

| | | | |
|---|---|---|---|
| **8 LM2T B6142** | Black | 10 | 0.037 |
| **8 LM2T B6143** | Green | 10 | 0.037 |
| **8 LM2T B6144** | Red | 10 | 0.037 |
| **8 LM2T B6344** | Red | 10 | 0.054 |

Ø40mm/1.6" (without mounting adapter).
For emergency stopping, ISO 13850 compliant.

| | | | |
|---|---|---|---|
| **8 LM2T B6644** | Red | 10 | 0.087 |

LATCH, TURN KEY TO RELEASE.
Ø40mm/1.6" (without mounting adapter).
For normal stopping.

| | | | |
|---|---|---|---|
| **8 LM2T B6542** | Black | 10 | 0.091 |
| **8 LM2T B6542G❶** | | 1 | 0.091 |
| **8 LM2T B6544** | Red | 10 | 0.091 |

**Operational characteristics**
- Any mounting position allowed
- Ambient conditions:
  - Operating temperature: -25...+60°C
  - Storage temperature: -40...+70°C
- Degree of protection:
  - Per IEC/EN: IP65
  - Per UL/CSA: Type 1, 2, 3R, 4, 4X, 12, 12K.

**Materials**
An aluminium and zinc alloy (zama) is used for the metal part whereas plastic parts are made of polyamide and polycarbonate.

**Mechanical endurance**
Operating force: 0.8kg/1.7lb (actuator)
Mechanical life: 1,000,000 cycles.

**Mounting adapter**

Type: LM2T AU120.
The adapter is fixed to the mounting surface by means of incorporated screws (Tmax = 0.8Nm/0.59lbft).
Actuators latch onto the mounting adapter by simple rotation through a Ø22mm/Ø0.87" drilling also on the cover of LPZ control stations.

**Contact elements for spring-return actuators with symbols**

Type: LM2T C10 (1NO)
LM2T CF10 (1NO Faston)
LM2T C10A (1EM)
LM2T C01 (1NC)
LM2T CF01 (1NC Faston)
LM2T C01D (1LB)

Contact elements snap onto the mounting adapter.
Up to 6 contacts can be fitted: 2 each on the left, middle and right, one behind the other.
Up to 3 contacts per actuator can be internally fitted on the cover surface of LPZ control stations.
To use contact elements in the middle position, install the action plug LM2T A140 on the actuator; see page 7-38.

**Certifications and compliance**
Certifications obtained: UL Listed for USA and Canada, (cULus - File E93601), as Auxiliary Devices; EAC, RINA, LROS, CCC.
Compliant with standards: IIEC/EN 60947-1, IEC/EN 60947-5-1, UL508, CSA C22.2 n° 14.

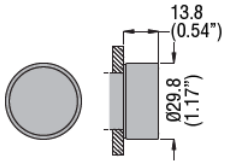
Lovato electric

ENERGY AND AUTOMATION

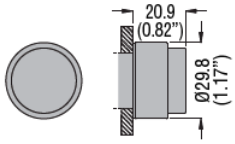
# LM2T METAL PUSHBUTTONS DATASHEET

| Order code | Description | Qty per pkg | Wt |
|---|---|---|---|
| | | n° | [kg] |
| Only for metal series LM2T actuators. | | | |
| **8 LM2T AU120** | Mounting adapter | 10 | 0.019 |

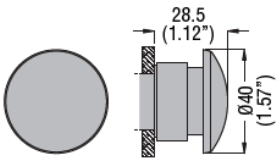
| Order code | Function | Qty per pkg | Wt |
|---|---|---|---|
| | | n° | [kg] |
| Screw termination. Without mounting adapter | | | |
| **8 LM2T C10❶** | .3 / .4 NO❶ | 10 | 0.011 |
| **8 LM2T C10A** | .7 / .8 EM❷ | 10 | 0.011 |
| **8 LM2T C01** ⊖ | .1 / ⊖ / .2 NC❸ | 10 | 0.011 |
| **8 LM2T C01D❶** | .5 / .6 LB❶❹ | 10 | 0.011 |



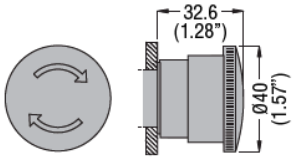

Flush
**LM2T B1...**



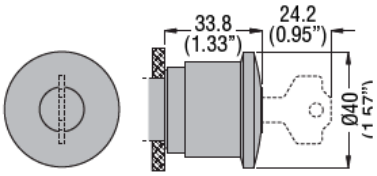

Extended
**LM2T B2...**



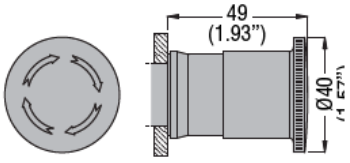

Spring return
**LM2T B614...**




Turn to release
**LM2T B634...**




Turn key to release
**LM2T B654...**




Turn to release
**LM2T B6644**

**Operational characteristics**
- Any mounting position allowed
- The adapter is fixed to the mounting surface by means of incorporated screws (Tmax = 0.8Nm/0.59lbft) also internally on LPZ control stations covers
- Contact elements snap onto the adapter
- Maximum tightening torque for screw terminals: 1Nm
- Ambient conditions:
  - Operating temperature: -25...+60°C
  - Storage temperature: -40...+70°C
- Degree of protection:
  - IP20 for contacts elements with screw termination
  - IP00 for contact elements with Faston termination.

**General characteristics of contact elements**

Wiping action and dual scraping and oscillating effect,
IEC rated insulation voltage: 690V
IEC rated thermal current Ith: 10A
Conductivity: 5V 10mA.
UL/CSA and IEC/EN 60947-5-1 designation: A600 Q600.
IEC/EN operational characteristics in AC15 category:

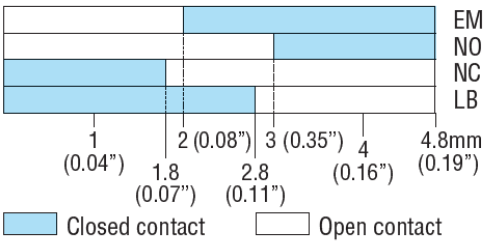
| [V] | 12 | 24 | 48 | 120 | 240 | 400 | 480 | 500 | 600 |
|---|---|---|---|---|---|---|---|---|---|
| [A] | 6 | 6 | 6 | 6 | 3 | 1.9 | 1.5 | 1.4 | 1.2 |

IEC/EN operational characteristics in DC13 category:

| [V] | 12 | 24 | 48 | 125 | 250 | 440 | 500 | 600 |
|---|---|---|---|---|---|---|---|---|
| [A] | 3 | 3 | 1.5 | 0.55 | 0.27 | 0.15 | 0.13 | 0.1 |

Short-circuit protection fuse: max. calibre 10A gG/SC
Contact resistance: ≤20mΩ
Terminals: Clamp screw with washer
Faston 1-6.35mm/0.25" or 2-2.8mm/0.11".

**Stroke of contact elements**




**Maximum conductor cross section for screw terminals** 1 or 2 2.5mm² or AWG12 cables.

**Mechanical and electric endurance**

Operating force: 0.5kg / 1.1lb (contact elements).
Electrical life:

| | |
|---|---|
| LM2T C10 | 1,000,000 cycles |
| LM2T CF10 | 1,000,000 cycles |
| LM2T C01 | 1,000,000 cycles |
| LM2T CF01 | 1,000,000 cycles |
| LM2T C10A | 600,000 cycles |
| LM2T C01D | 600,000 cycles. |

**Certifications and compliance**

Certifications obtained: RINA, LROS, EAC, CCC, UL Listed, for USA and Canada (cULus - File E93601), as Auxiliary Devices.
Compliant with standards: IEC/EN 60947-1, IEC/EN 60947-5-1, UL508, CSA C22.2 n° 14.

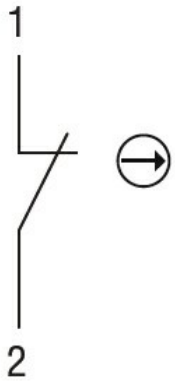
# 8LM2TC01

CONTACT ELEMENT, Ø22MM 8LM METAL SERIES, NC. SCREW TERMINATION

| | | | |
|---|---|---|---|
| Product designation | | | Contact element - Screw termination |
| Product type designation | | | 8LM2TC01 |
| **Electrical characteristics** | | | |
| Rated insulation voltage Ui IEC/EN | | V | 690 |
| IEC Conventional free air thermal current Ith | | A | 10 |
| Conductivity | | | 5V 10mA |
| UL/CSA and IEC/EN 60947-5-1 designation | | | A600 Q600 |
| Operating current AC15 | | | |
| | 12V | A | 6 |
| | 24V | A | 6 |
| | 48V | A | 6 |
| | 120V | A | 6 |
| | 240V | A | 3 |
| | 400V | A | 1.9 |
| | 480V | A | 1.5 |
| | 500V | A | 1.4 |
| | 600V | A | 1.2 |
| Operating current DC13 | | | |
| | 12V | A | 3 |
| | 24V | A | 3 |
| | 48V | A | 1.5 |
| | 125V | A | 0.55 |
| | 250V | A | 0.27 |
| | 440V | A | 0.15 |
| | 500V | A | 0.13 |
| | 600V | A | 0.1 |
| Contact resistance | | mΩ | ≤20 |
| **Mechanical features** | | | |
| Tightening torque for terminals max | | Nm | 1 |
| Terminals screw | | | Clamp screw with washer |
| Conductor section | | | |
| AWG/kcmil conductor section | | | |
| | max | | 12 |
| IEC | | | |
| | max | mm² | 1 or 2 2.5 |
| Weight | | g | 11 |
| **Ambient conditions** | | | |
| Temperature | | | |
| Operating temperature | | | |
| | min | °C | -25 |
| | max | °C | +60 |
| Storage temperature | | | |
| | min | °C | -40 |

| | | | |
|---|---|---|---|
| max | °C | +70 | |

## Wiring diagrams




## Certifications and compliance

Compliance

| |
|---|
| CSA C22.2 n° 14 |
| IEC/EN 60947-1 |
| IEC/EN 60947-5-1 |
| UL508 |

Certificates

| |
|---|
| CCC |
| cULus |
| EAC |
| LROS |
| RINA |
| UL |

# Bibliography


[1] FISCHERTECHNIK (Unknown), «3D-Robot 24V - Simulation - fischertechnik», https://www.fischertechnik.de/en/products/simulating/training-models/511938-sim-3d-robot-24v-simulation, (Accessed on 06/21/2021). (Cited at page 57)

[2] HEGAMURL (2019), «TIA Portal: Open User Communication TSEND + TRCV / PLC-PLC Communication - YouTube», https://www.youtube.com/watch?v=fVLVvqn03qM, (Accessed on 06/22/2021). (Cited at page 70)

[3] MATHWORKS (????), «MATLAB Documentation - MathWorks Italia», https://it.mathworks.com/help/matlab/, (Accessed on 06/22/2021). (Cited at page 9)

[4] NINJA, M. (2018), «Vrep (CoppeliaSim) Inverse kinematics tutorial (before version 4.2.0) - YouTube», https://www.youtube.com/watch?v=eTd6mOk6Njw, (Accessed on 06/22/2021). (Cited at page 65)

[5] ROBOTICS, C. (Unknown), «Robot simulator CoppeliaSim: create, compose, simulate, any robot - Coppelia Robotics», https://www.coppeliarobotics.com/, (Accessed on 06/22/2021). (Cited at page 9)

[6] VERMA, H. (2021), «Path Following Robot In V-REP/ Coppeliasim Tutorial | Drone | Robot Arm - YouTube», https://www.youtube.com/watch?v=9Epp4C6xHF4, (Accessed on 06/22/2021). (Cited at page 65)

[7] WIKIPEDIA (Unknown), «CoppeliaSim - Wikipedia», `https://en.wikipedia.org/wiki/CoppeliaSim`, (Accessed on 06/22/2021). (Cited at page 9)