# PLC Configuration Document

## Overview

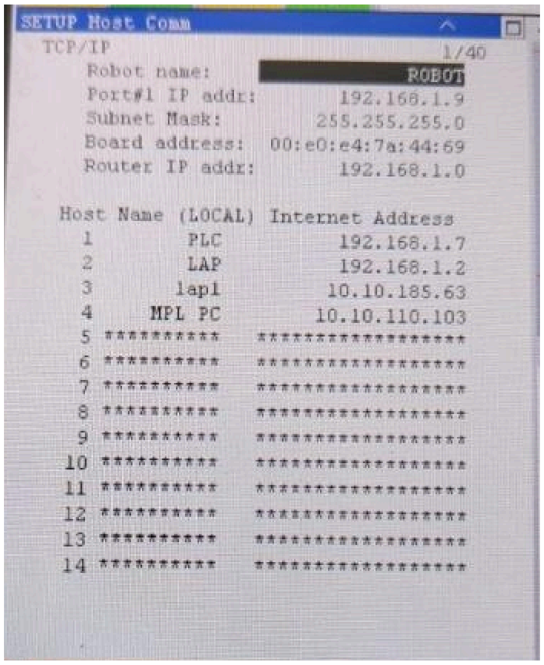
This project focuses on achieving autonomous control of the FANUC LR Mate 200iD/4S industrial robot by integrating Python-based kinematic computation, PLC-driven I/O mapping, and robot-side execution logic. Using Ethernet/IP, the Allen-Bradley CompactLogix PLC acts as a bridge between Python and the robot, with virtual tags enabling seamless data exchange. Python handles inverse kinematics using the Robotics Toolbox to convert user-defined positions into joint angles, which are then transmitted to the PLC. The PLC performs int-to-bit conversion and maps these values to the robot's I/O. On the robot side, Group I/O instructions convert incoming digital inputs back into joint values and assign them to Position Registers for motion execution. Concurrently, the robot runs background logic that reads its current joint positions, sends them back through group outputs, and allows Python to compare feedback with target values to decide whether to proceed. A key objective of this project is to achieve complete control without relying on FANUC's proprietary motion control or expensive core packages, instead opting for an open-source, externally-driven control framework. Initial experiments with camera-based vision calibration were also conducted, laying the groundwork for future enhancements in visual servoing and environment-aware motion.

## ROBOT CONFIGURATION

The robot's configuration ensures reliable communication and accurate data processing for seamless operation with the PLC and Robot. This section outlines the necessary setup steps to enable joint value transfer and control through Ethernet/IP and Group I/O.

### STEP 1: Host Communication Setup

To establish basic communication, navigate to **MENU → SETUP → Host Comm → TCP/IP** and configure the robot's IP address, device name, and subnet mask. Add the PLC and the Python control system as remote hosts to enable proper network-level data exchange between the robot and external systems.

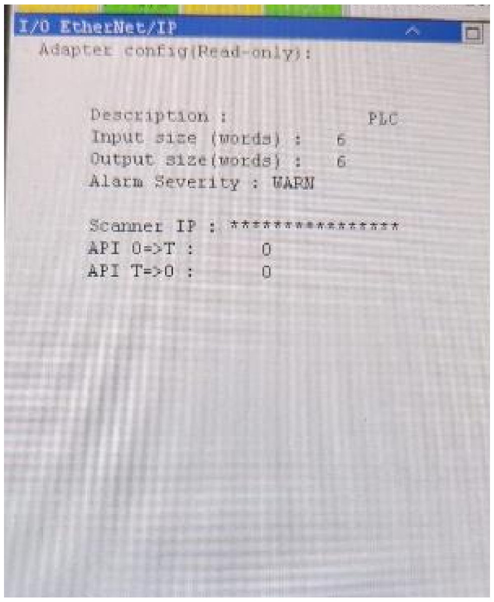


```
SETUP Host Comm
  TCP/IP                                        1/40
      Robot name:                              ROBOT
      Port#1 IP addr:                    192.168.1.9
      Subnet Mask:                     255.255.255.0
      Board address:               00:e0:e4:7a:44:69
      Router IP addr:                    192.168.1.0

    Host Name (LOCAL)  Internet Address
      1          PLC                     192.168.1.7
      2          LAP                     192.168.1.2
      3         lap1                   10.10.185.63
      4       MPL PC                  10.10.110.103
      5 **********    *******************
      6 **********    *******************
      7 **********    *******************
      8 **********    *******************
      9 **********    *******************
     10 **********    *******************
     11 **********    *******************
     12 **********    *******************
     13 **********    *******************
     14 **********    *******************
```

**STEP 2: Ethernet/IP Configuration**

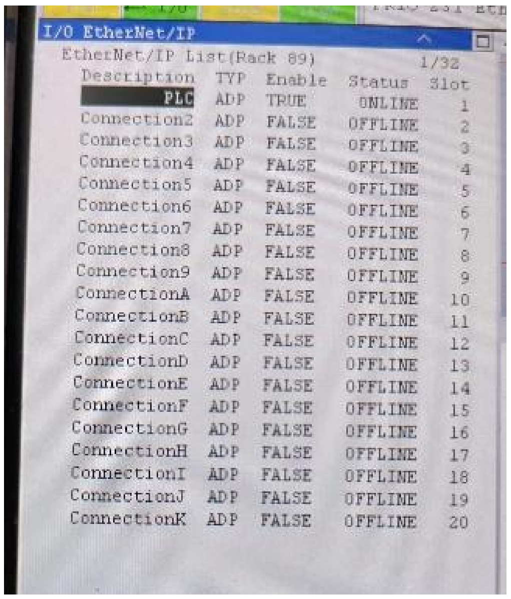
Ethernet/IP communication is set up by going to **MENU → I/O → EtherNet/IP**, where you configure the input and output sizes. Since each of the six joints requires 16 bits, a total of 96 I/Os are needed. Set the input and output sizes to 6 words each to cover this requirement. Once configured correctly, ensure that the connection status shows as Online, confirming that the robot is communicating with the PLC.



```
I/O EtherNet/IP
  Adapter config(Read-only):


      Description :                PLC
      Input size (words) :   6
      Output size(words) :   6
      Alarm Severity : WARN

      Scanner IP : ****************
      API O=>T :         0
      API T=>O :         0
```

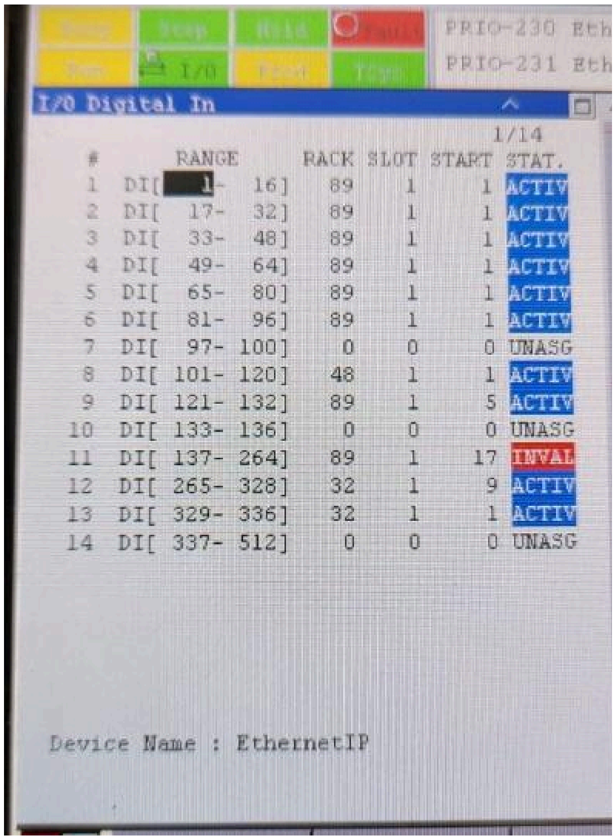


```
I/O EtherNet/IP
  EtherNet/IP List(Rack 89)                    1/32
    Description   TYP  Enable   Status   Slot
            PLC   ADP  TRUE     ONLINE     1
    Connection2   ADP  FALSE    OFFLINE    2
    Connection3   ADP  FALSE    OFFLINE    3
    Connection4   ADP  FALSE    OFFLINE    4
    Connection5   ADP  FALSE    OFFLINE    5
    Connection6   ADP  FALSE    OFFLINE    6
    Connection7   ADP  FALSE    OFFLINE    7
    Connection8   ADP  FALSE    OFFLINE    8
    Connection9   ADP  FALSE    OFFLINE    9
    ConnectionA   ADP  FALSE    OFFLINE   10
    ConnectionB   ADP  FALSE    OFFLINE   11
    ConnectionC   ADP  FALSE    OFFLINE   12
    ConnectionD   ADP  FALSE    OFFLINE   13
    ConnectionE   ADP  FALSE    OFFLINE   14
    ConnectionF   ADP  FALSE    OFFLINE   15
    ConnectionG   ADP  FALSE    OFFLINE   16
    ConnectionH   ADP  FALSE    OFFLINE   17
    ConnectionI   ADP  FALSE    OFFLINE   18
    ConnectionJ   ADP  FALSE    OFFLINE   19
    ConnectionK   ADP  FALSE    OFFLINE   20
```

**Note:** 1 Joint Position = 2 bytes = 16 bits.

**STEP 3: Digital I/O Configuration**

Next, digital I/O ranges are assigned via **MENU → I/O → Digital → Configure**. For each joint, allocate a range of 16 digital input bits, using Rack: 89, Slot: 1, and the appropriate starting bit. This setup should be repeated across all six joints to fully define the 96-bit input space.



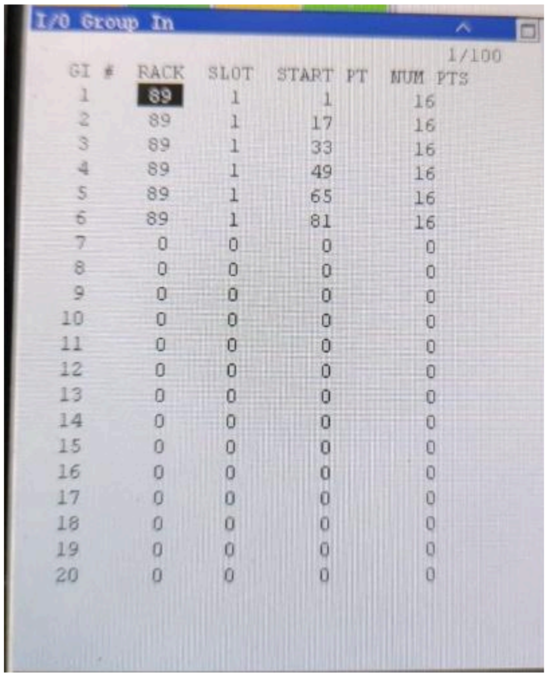
```
I/O Digital In
                                        1/14
 #        RANGE        RACK SLOT START STAT.
 1  DI[    1-   16]     89    1     1 ACTIV
 2  DI[   17-   32]     89    1     1 ACTIV
 3  DI[   33-   48]     89    1     1 ACTIV
 4  DI[   49-   64]     89    1     1 ACTIV
 5  DI[   65-   80]     89    1     1 ACTIV
 6  DI[   81-   96]     89    1     1 ACTIV
 7  DI[   97-  100]      0    0     0 UNASG
 8  DI[  101-  120]     48    1     1 ACTIV
 9  DI[  121-  132]     89    1     5 ACTIV
10  DI[  133-  136]      0    0     0 UNASG
11  DI[  137-  264]     89    1    17 INVAL
12  DI[  265-  328]     32    1     9 ACTIV
13  DI[  329-  336]     32    1     1 ACTIV
14  DI[  337-  512]      0    0     0 UNASG

Device Name : EthernetIP
```

**Note:** We don't need to configure the digital outputs on the robot side. The PLC can directly read the robot's outputs, which can then be accessed through Python. This process is explained in detail in the Virtual Tag Creation section.

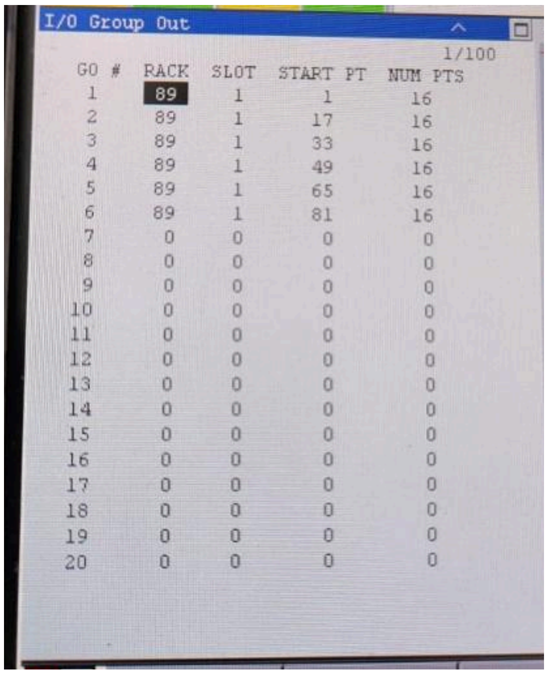
**STEP 4: Group I/O Configuration**

Group I/O simplifies the handling of multiple digital bits by converting them into integer values. Go to **MENU → I/O → Group → Configure**, and for each joint, set Rack: 89, Slot: 1, and Start as the starting bit of each grouping. Assign 16 bits per group to correspond with each joint's data. This configuration allows the robot to interpret incoming I/O as integer values and store them in position registers.


I/O Group In

1/100

| GI # | RACK | SLOT | START PT | NUM PTS |
|---|---|---|---|---|
| 1 | 89 | 1 | 1 | 16 |
| 2 | 89 | 1 | 17 | 16 |
| 3 | 89 | 1 | 33 | 16 |
| 4 | 89 | 1 | 49 | 16 |
| 5 | 89 | 1 | 65 | 16 |
| 6 | 89 | 1 | 81 | 16 |
| 7 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 |
| 9 | 0 | 0 | 0 | 0 |
| 10 | 0 | 0 | 0 | 0 |
| 11 | 0 | 0 | 0 | 0 |
| 12 | 0 | 0 | 0 | 0 |
| 13 | 0 | 0 | 0 | 0 |
| 14 | 0 | 0 | 0 | 0 |
| 15 | 0 | 0 | 0 | 0 |
| 16 | 0 | 0 | 0 | 0 |
| 17 | 0 | 0 | 0 | 0 |
| 18 | 0 | 0 | 0 | 0 |
| 19 | 0 | 0 | 0 | 0 |
| 20 | 0 | 0 | 0 | 0 |


I/O Group Out

1/100

| GO # | RACK | SLOT | START PT | NUM PTS |
|---|---|---|---|---|
| 1 | 89 | 1 | 1 | 16 |
| 2 | 89 | 1 | 17 | 16 |
| 3 | 89 | 1 | 33 | 16 |
| 4 | 89 | 1 | 49 | 16 |
| 5 | 89 | 1 | 65 | 16 |
| 6 | 89 | 1 | 81 | 16 |
| 7 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 |
| 9 | 0 | 0 | 0 | 0 |
| 10 | 0 | 0 | 0 | 0 |
| 11 | 0 | 0 | 0 | 0 |
| 12 | 0 | 0 | 0 | 0 |
| 13 | 0 | 0 | 0 | 0 |
| 14 | 0 | 0 | 0 | 0 |
| 15 | 0 | 0 | 0 | 0 |
| 16 | 0 | 0 | 0 | 0 |
| 17 | 0 | 0 | 0 | 0 |
| 18 | 0 | 0 | 0 | 0 |
| 19 | 0 | 0 | 0 | 0 |
| 20 | 0 | 0 | 0 | 0 |

**STEP 5: Final Verification**

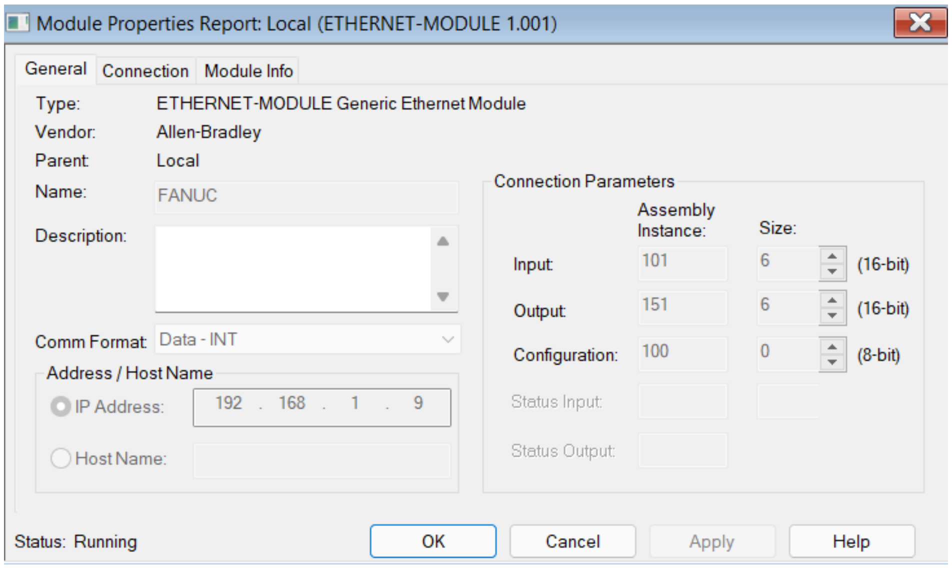
After all configurations are completed, restart the robot controller to apply the changes. Open the Digital I/O Monitor and Group I/O Monitor pages to verify that the I/Os are active, correctly mapped, and responding. This ensures proper signal flow before starting integration with the PLC and Python-based control loop.

## PLC CONFIGURATION

The PLC acts as a communication bridge between Python and the robot, handling data conversion, I/O mapping, and motion triggering logic. The steps below detail how the FANUC robot is integrated with the Allen-Bradley CompactLogix PLC.

**STEP 1: Adding the FANUC Robot Module in PLC**

In the PLC software (Studio 5000, used for this project), add the robot using Ethernet/IP communication. Search for the module type “Generic Ethernet Module”, which allows flexible configuration of custom I/O sizes. While a predefined module named “EtherNet/IP Robot” exists, it is not suitable for our use case. Enter the robot’s IP address and module name to establish communication. Make sure to set the Comm Format to INT and configure the input/output sizes according to the I/O setup done on the robot side.



Module Properties Report: Local (ETHERNET-MODULE 1.001)

General | Connection | Module Info

| | |
|---|---|
| Type: | ETHERNET-MODULE Generic Ethernet Module |
| Vendor: | Allen-Bradley |
| Parent: | Local |
| Name: | FANUC |
| Description: | |
| Comm Format: | Data - INT |

Address / Host Name

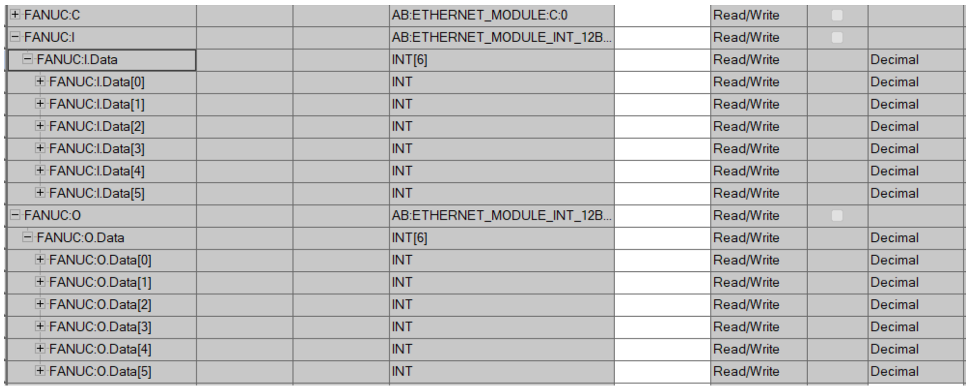
| | |
|---|---|
| IP Address: | 192 . 168 . 1 . 9 |
| Host Name: | |

Connection Parameters

| | Assembly Instance: | Size: | |
|---|---|---|---|
| Input: | 101 | 6 | (16-bit) |
| Output: | 151 | 6 | (16-bit) |
| Configuration: | 100 | 0 | (8-bit) |
| Status Input: | | | |
| Status Output: | | | |

Status: Running

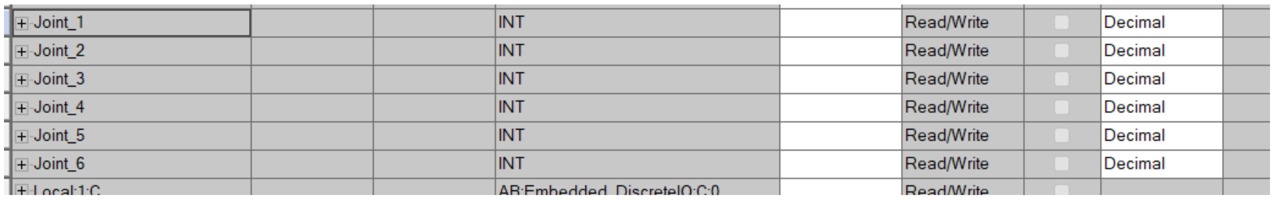
OK | Cancel | Apply | Help



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| ⊞ FANUC:C | | | AB:ETHERNET_MODULE:C:0 | | Read/Write | ☐ | |
| ⊟ FANUC:I | | | AB:ETHERNET_MODULE_INT_12B... | | Read/Write | ☐ | |
| ⊟ FANUC:I.Data | | | INT[6] | | Read/Write | | Decimal |
| ⊞ FANUC:I.Data[0] | | | INT | | Read/Write | | Decimal |
| ⊞ FANUC:I.Data[1] | | | INT | | Read/Write | | Decimal |
| ⊞ FANUC:I.Data[2] | | | INT | | Read/Write | | Decimal |
| ⊞ FANUC:I.Data[3] | | | INT | | Read/Write | | Decimal |
| ⊞ FANUC:I.Data[4] | | | INT | | Read/Write | | Decimal |
| ⊞ FANUC:I.Data[5] | | | INT | | Read/Write | | Decimal |
| ⊟ FANUC:O | | | AB:ETHERNET_MODULE_INT_12B... | | Read/Write | ☐ | |
| ⊟ FANUC:O.Data | | | INT[6] | | Read/Write | | Decimal |
| ⊞ FANUC:O.Data[0] | | | INT | | Read/Write | | Decimal |
| ⊞ FANUC:O.Data[1] | | | INT | | Read/Write | | Decimal |
| ⊞ FANUC:O.Data[2] | | | INT | | Read/Write | | Decimal |
| ⊞ FANUC:O.Data[3] | | | INT | | Read/Write | | Decimal |
| ⊞ FANUC:O.Data[4] | | | INT | | Read/Write | | Decimal |
| ⊞ FANUC:O.Data[5] | | | INT | | Read/Write | | Decimal |

**STEP 2: Virtual Tag Creation**

Virtual tags are created to control the robot's digital inputs (outputs from the PLC). These tags are used by Python to send joint values. Create input tags in the PLC that match the robot's configured word size — each tag representing one word (2 bytes). A total of 6 words (96 bits) are used to represent all six joints.

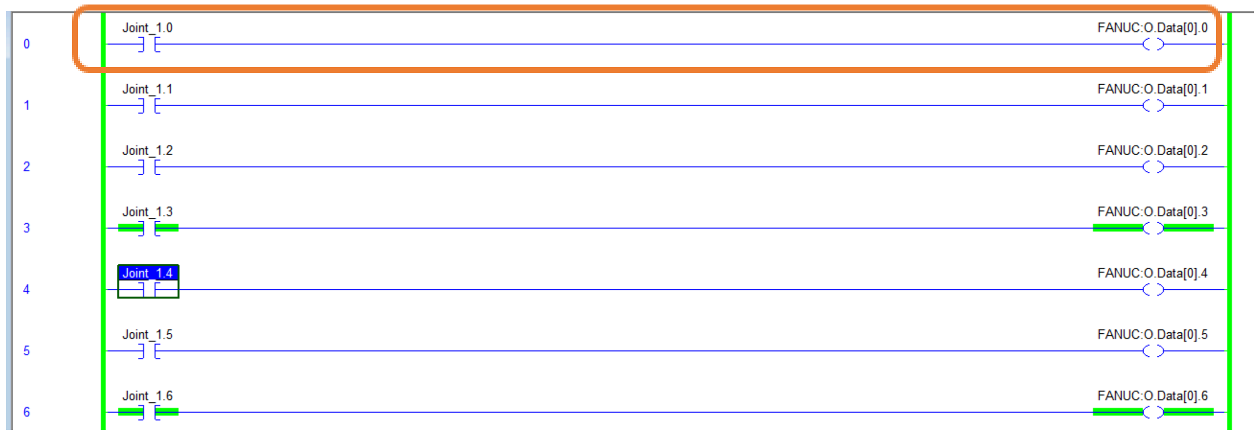


| | | | | | | | |
|---|---|---|---|---|---|---|---|
| ⊞ Joint_1 | | | INT | | Read/Write | ☐ | Decimal |
| ⊞ Joint_2 | | | INT | | Read/Write | ☐ | Decimal |
| ⊞ Joint_3 | | | INT | | Read/Write | ☐ | Decimal |
| ⊞ Joint_4 | | | INT | | Read/Write | ☐ | Decimal |
| ⊞ Joint_5 | | | INT | | Read/Write | ☐ | Decimal |
| ⊞ Joint_6 | | | INT | | Read/Write | ☐ | Decimal |
| ⊞ Local:1:C | | | AB:Embedded_DiscreteIO:C:0 | | Read/Write | ☐ | |

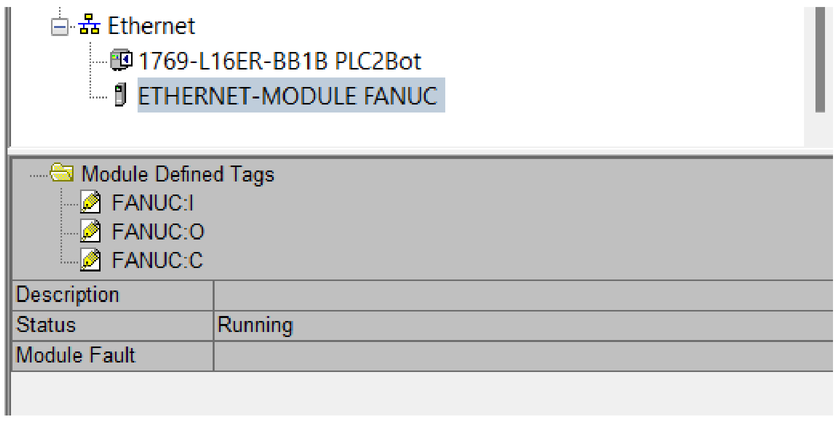
**Note:** The robot does not require configuration for digital outputs, as its output values can be directly read in the PLC and accessed via Python. Therefore, feedback from the robot does not need virtual tag creation or bitwise mapping — the PLC can directly access robot output data (as digital inputs in the PLC).

### STEP 3: Ladder Logic Programming for Data Processing

Develop ladder logic to map individual bits from the virtual tags to specific robot output bits. For example, for Joint 1, map bit 0 from the virtual tag to robot output 0.0. Repeat this mapping for all six joints, ensuring 16 bits per joint, totaling 96 bits. This allows Python to send joint data in word format, which is split into bits by the PLC and passed to the robot's I/O system.



### STEP 4: Final Verification



| | |
|---|---|
| Description | |
| Status | Running |
| Module Fault | |

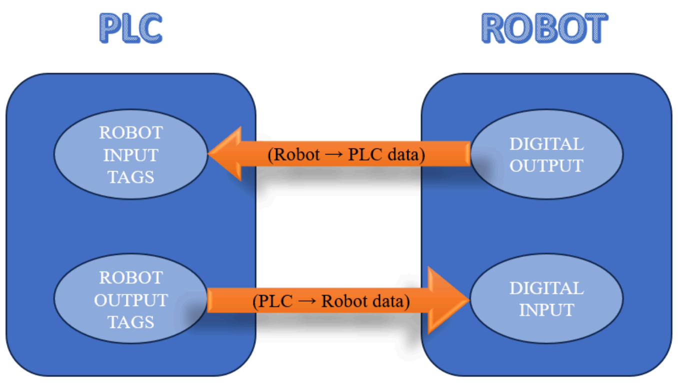
Connect to the PLC and go online with the controller. Switch the PLC to Run Mode and observe the module status. If no faults or connection errors appear, the PLC is successfully communicating with the robot and ready for operation through Python.

## ROBOT PROGRAMMING

Robot programs are designed to process movement data and provide feedback.

**Creating TP Programs for Motion Execution**

The following programs facilitate robot movement and data exchange:



1. **Reading Data from PLC:**
   - Read input values from the PLC.
   - Convert the bit data into INT format using Group Input.
   - Assign the processed values to Position Registers (PRs).
   - Then add the line of the execution program and run it in a loop.
2. **Providing Robot Position Feedback:**
   - Read the robot’s current position via system variables.
   - Store the values in Position Registers.
   - Send the position data back to the PLC via Group Output.
   - Run this process as a background logic task to ensure continuous updates.