Engineering Department
University of Messina

Master's Degree Course in Electronic Engineering for Industry
Industrial Automation

Speech Recognition System to control a Robotic Arm

WRITTEN BY:

ALESSANDRO FICARRA
ROBERTO CARDILLO
GABRIELE IELO
GABRIELE RUGGERI

PROFESSOR:

PROF. L. PATANÉ

Academic year 2022-2023

Contents

CHAPTER 1

Introduction

We chose to work on the development of a Google-based Speech Recognition system aimed towards moving and controlling a Fischertechnick robotic arm. The microphone used is an everyday microphone, making the software accessible to a large demographic, and, thanks to some Python libraries, the voice recorded gets recognized and transformed into a command message, promptly sent to a Matlab script connected to the PLC with a TCP connection. The PLC is managed thanks to the software named Tia Portal. The hardware part and the various link settings were taken from the "Gesture Detection System to control a Robotic Arm" project.[1]

CHAPTER 2

Hardware aspects

This chapter will be covering all the aspects related to the hardware parts used for building the robotic arm, in particular the device used and their characteristics are described. Though the realized project is focused only on the development of the gesture detection system, the hardware part of the robotic harm has been also documented for completeness.

2.1 PLC SIMATIC S7-1200

The PLC SIMATIC S7-1200 with CPU 1215C AC/DC/RLY (Figure 2.1) is a PLC produced by Siemens. It is used in the project to realize the sequence control of the robotic arm.
It is characterized by:

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

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

2.2 Fischertechnik 3D-Robot 24V

The 3D-Robot 24V (Figure 2.2) is a robotic arm produced by Fischertechnik. It is a 3-DOF (3 degrees of freedom) robot and in particular it is composed by:

- 1 revolute joint;
- 2 prismatic joints (up/down and forward/backward);
- 1 gripper as end-effector.

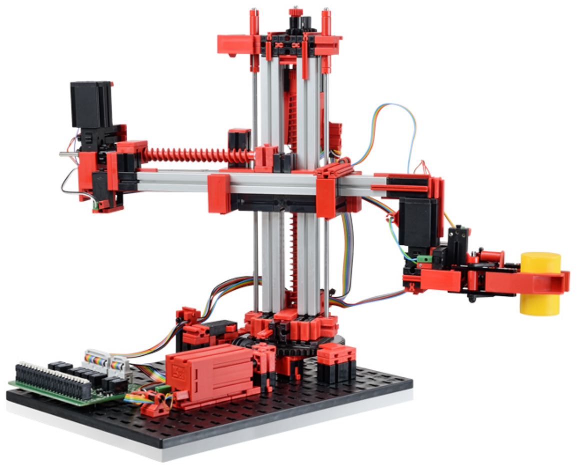


Figure 2.2: Fischertechnik 3D-Robot 24V

The robot is characterized by a cylindrical geometry, thus is workspace is a portion of a hollow cylinder.

2.2.1 Mini switches

The robotic arm as four mini switches (figure 2.3) 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.

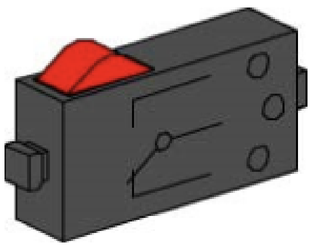


Figure 2.3: Mini switch

2.2.2 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.

2.2.3 Motor encoders

The robot has two motor encoders (figure 2.4) with a maximum frequency of 1 KHz, that can be used to understand respectively:

- how much end horizontal arm is moved down;
- 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 (figure 2.5).

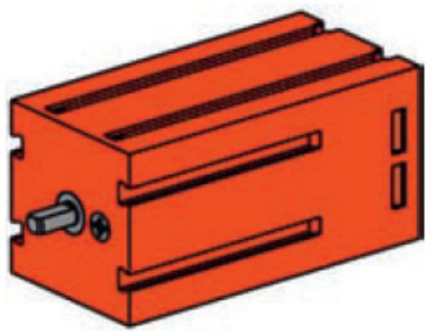


Figure 2.4: Motor encoder

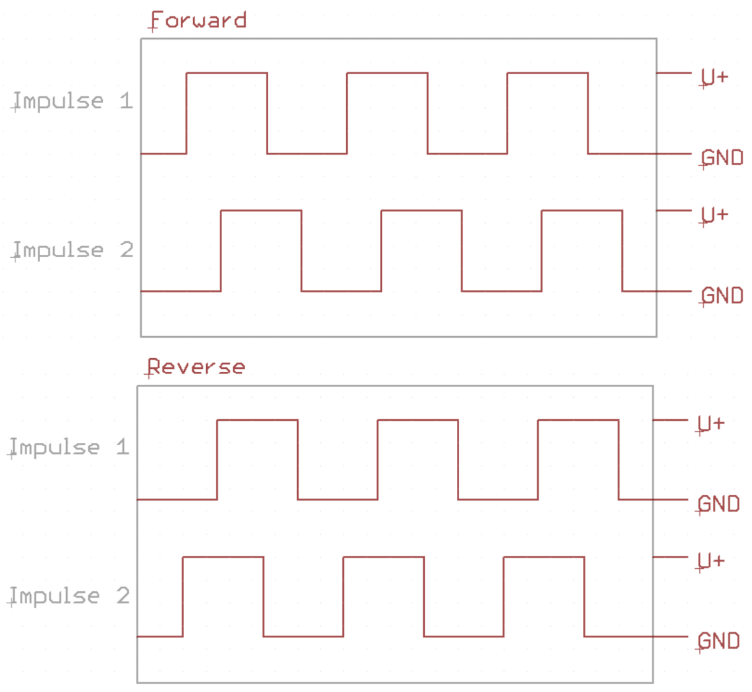
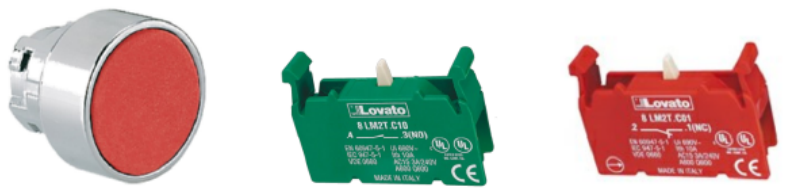


Figure 2.5: Motor encoder signals

2.2.4 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.

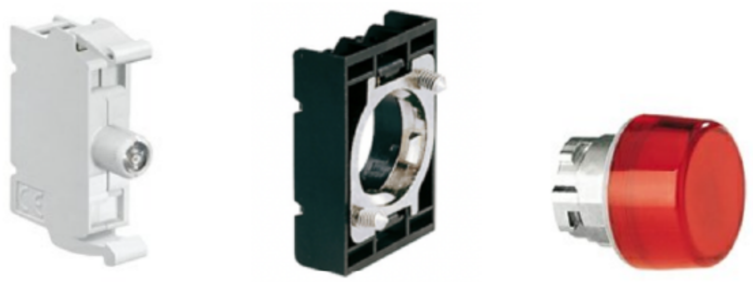


(a) Push-button **(b)** Normally open contact element **(c)** Normally closed contact element

Figure 2.6: Push buttons

2.2.5 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: LEDs

CHAPTER 3

Software aspects

This chapter will be presenting all the software used to develop the project.

3.1 TIA Portal

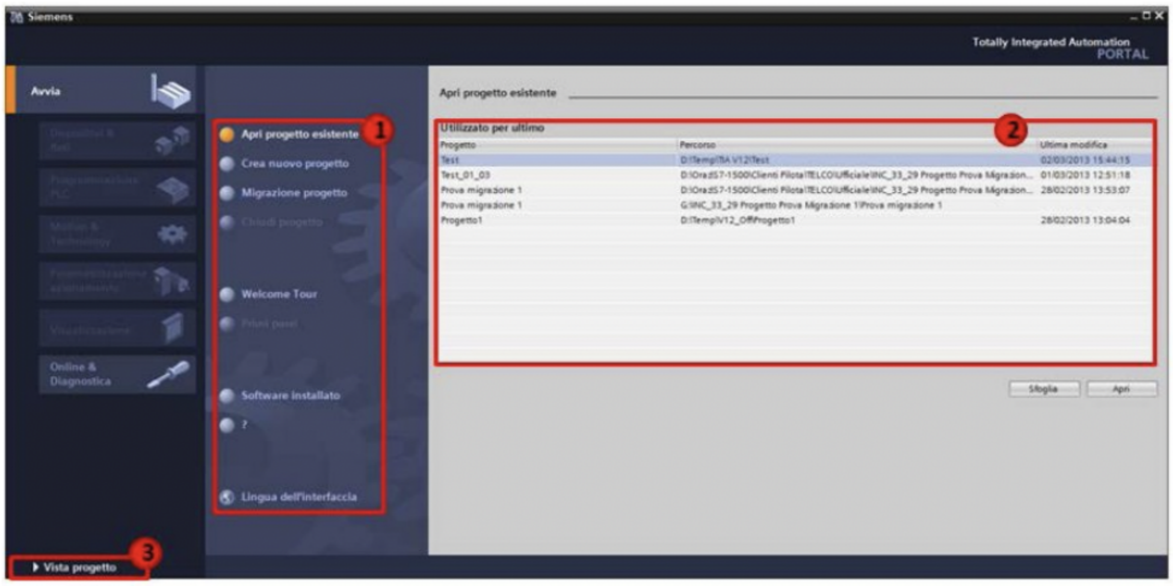
TIA Portal (Totally Integrated Automation Portal) is a software package by Siemens created specifically to develop automation using Siemens products such as PLCs. It is in practice a centralized design environment characterized by a common user interface for all automation tasks with shared services (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;

3.1.1 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.



Apri progetto esistente

Utilizzato per ultimo

Progetto	Percorso	Ultima modifica
Test	D:\Temp\TIA V12\Test	02/03/2013 15:44:15
Test_01_03	D:\Ora\S7-1500\Clienti Pilota\TELCO\Ufficiale\INC_33_29 Progetto Prova Migrazion...	01/03/2013 12:51:18
Prova migrazione 1	D:\Ora\S7-1500\Clienti Pilota\TELCO\Ufficiale\INC_33_29 Progetto Prova Migrazion...	28/02/2013 13:53:07
Prova migrazione 1	G:\INC_33_29 Progetto Prova Migrazione 1\Prova migrazione 1	
Progetto1	D:\Temp\V12_Off\Progetto1	28/02/2013 13:04:04

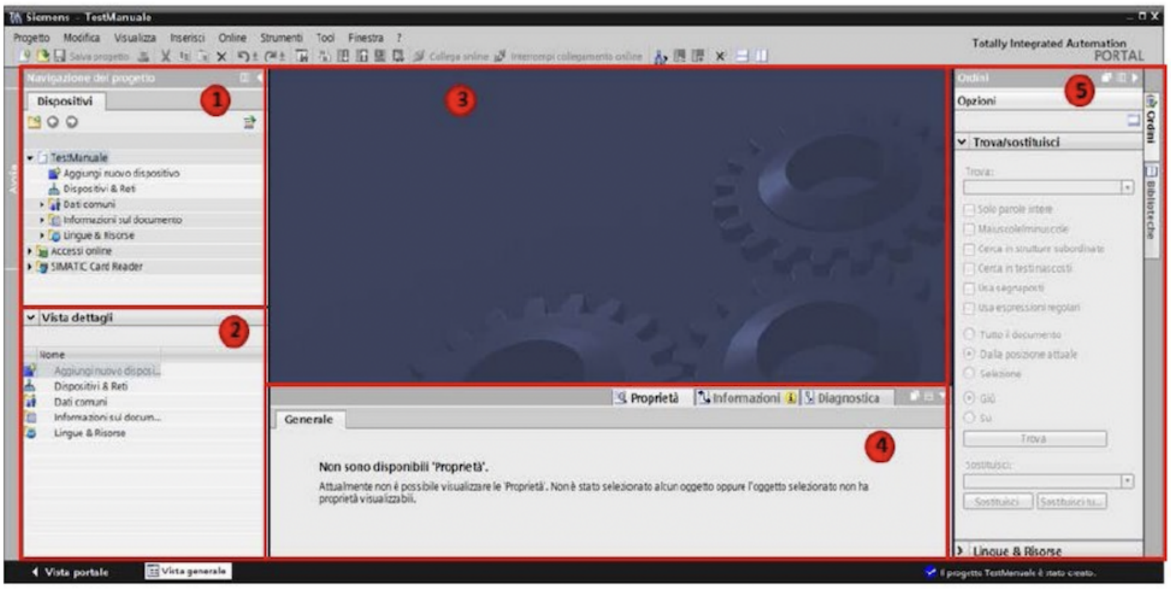
Figure 3.1: TIA Portal - portal view[1]

3.1.2 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.



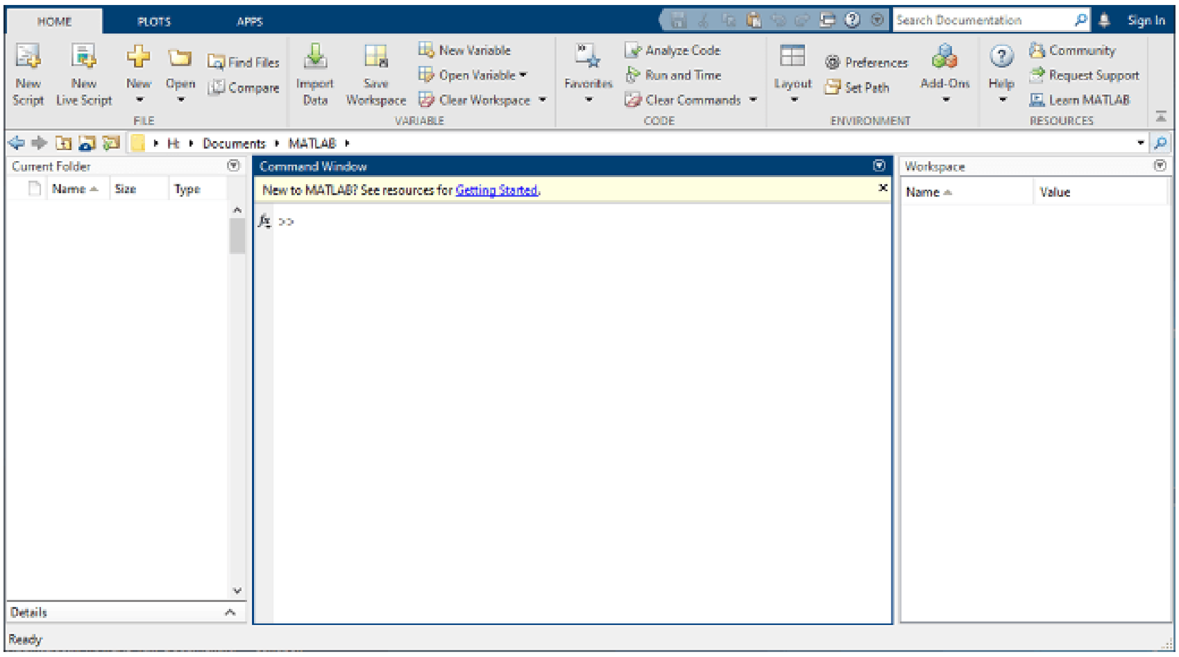
Non sono disponibili 'Proprietà'.
Attualmente non è possibile visualizzare le 'Proprietà'. Non è stato selezionato alcun oggetto oppure l'oggetto selezionato non ha proprietà visualizzabili.

Figure 3.2: TIA Portal - project view [1]

3.2 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 interface appears in its default layout. The interface includes these panels:

- Current Folder: that allows to access files in current working folder;
- Command Window: for entering commands in the command line, indicated by the prompt (»);
- Workspace: for exploring variables created or imported from files.



New to MATLAB? See resources for Getting Started.

```
>>
```

Figure 3.3: Matlab interface

Matlab is used in the project for the creation of a TCP connection between the laboratory computer and the PLC device. It is also used to create a TCP server at which the user computer can be connected as client in order to send the commands to move the robotic arm.

3.3 Python

python™

Figure 3.4: Python logo

Python is an object-oriented programming language suitable for application development, scripting, numerical computing and system testing. In this project, Python will be used to transform the voice picked up by the microphone into text and, based upon a user-defined dictionary, the text itself into a command. The library used to complete this task is the Speech_Recognition library based on Google; the Python version required to run this program is 3.8+.

CHAPTER 4

Implementation

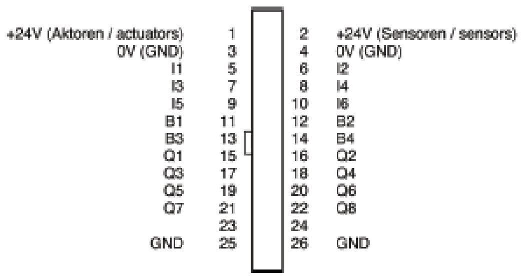
In this chapter all the implementations aspects are described in details.

4.1 Robotic arm

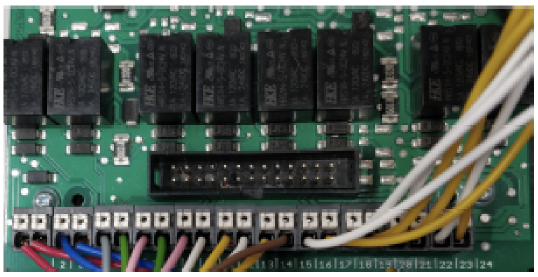
The hardware part related to the robotic arm is presented for completeness but were not part of this development activity, thus it is taken from the documentation of the robotic arm, developed by another team of students [2].

4.1.1 Hardware setup

In order to control the robotic arm, it's necessary to use its pinout of 24 pins (figure 4.1) with inputs and outputs. Each of these pins are related to a specific part of the robot and they can be used to control its movements. [1]
A more detailed description of the role of each pin is presented in figure 4.2. In which they are characterized by position, description, name, and the type (input or output).
This table is important because from it, it is 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.



(a) Robot Pinout



(b) Photo of Robot Pinout

Figure 4.1: Robot pinout [1]

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 4.2: Robot pinout description [1]

These pins are connected to the PLC, that is the control unit that will control the robotic arm movements. 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 figura 4.3 realized with KiCad, a free software suite for electronic design automation (EDA).

In the schema are also connected 3 buttons for emergency stop, robotic arm position reset and for establishing the connection with the laboratory computer, and 2 LEDs, one red for signaling the emergency stop and one green for signaling when the robot is moving or ready to receive commands.

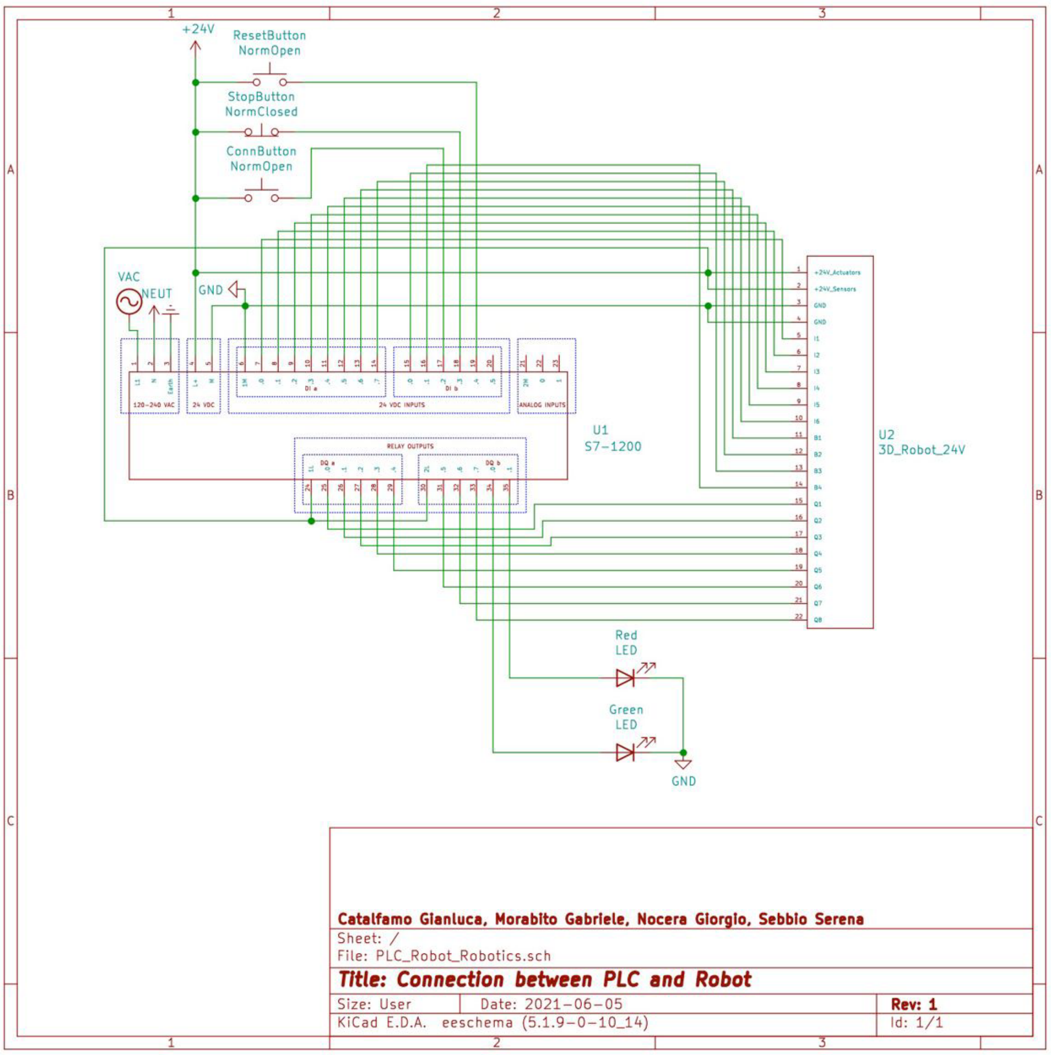


Figure 4.3: Connections between Robot and PLC [1]

4.2 Connection with the robotic arm

In this section it is presented the part related to the connection between the user computer and the PLC that controls the robotic arm. In figure 4.4 it is possible to see the schema of the project connections architecture.

TIA Portal

In order to allow the PLC to deal with a TCP connection, the first thing to do is to add to the ladder diagram the specific blocks 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

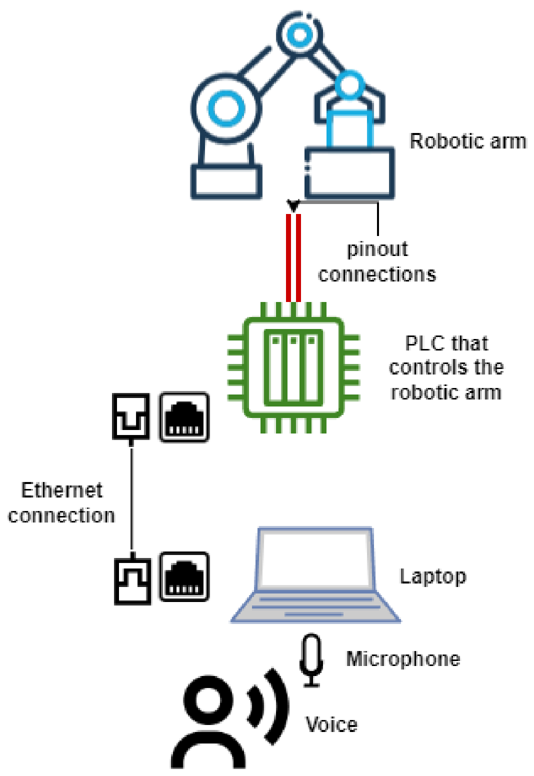


Figure 4.4: Project connections architecture

segment is added for each of the following blocks:

- **TCON** (figura 4.5). 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.

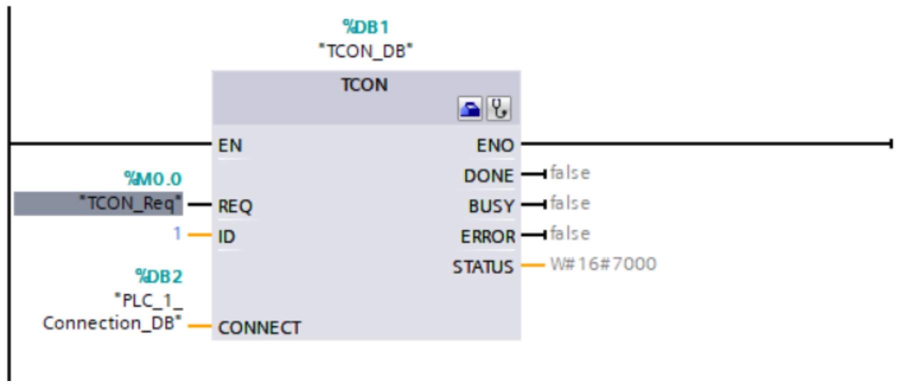


Figure 4.5: TCON function for the TCP connection [1]

- **TDISCON** (figure 4.6). 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.

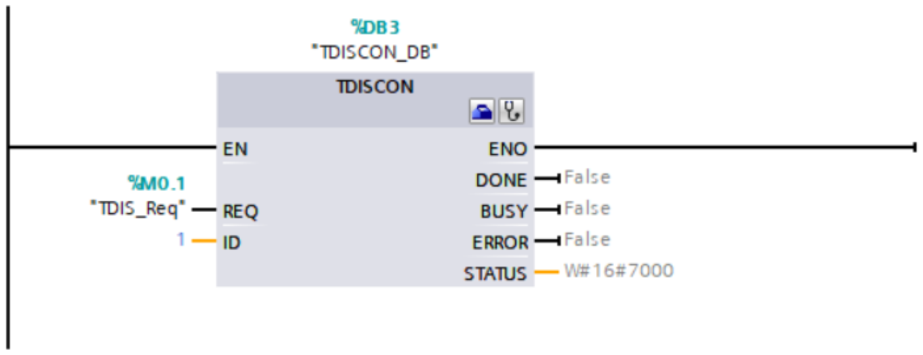


Figure 4.6: TDISCON function for the TCP connection [1]

- **TSEND** (figure 4.7). 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.

- **TRCV** (figure 4.8). 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

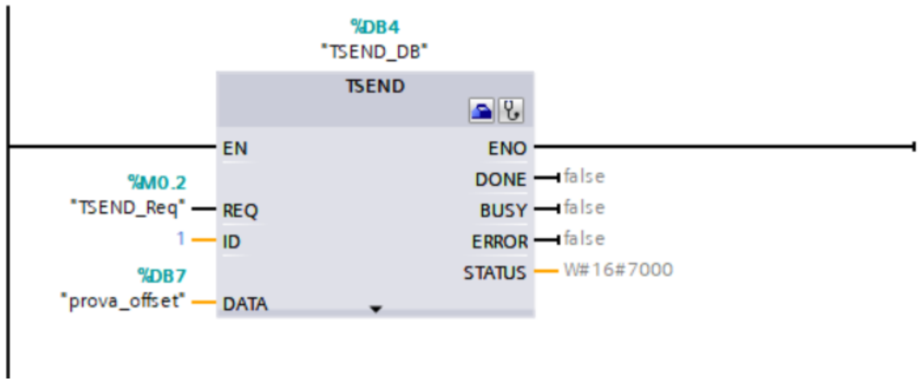


Figure 4.7: TSEND function for the TCP connection [1]

REQ input port. The input port "DATA" of this function is the place where the received data from the PLC is stored.

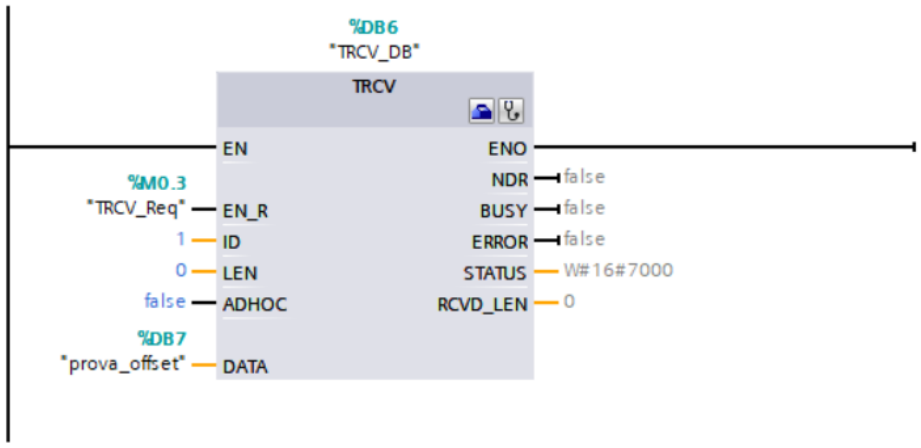


Figure 4.8: TRCV function for the TCP connection [1]

Matlab

In order to setup a TCP connection with Matlab, the first thing to do is to create a connection object. This can be done using the **tcpip** function, that is part of the Instrument Control Toolbox. It takes as first parameter the IP address of the partner (i.e. the PLC) and 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 network 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 the Matlab file "Matlab_PLC_connection.m" (at line 5):

```
global t;
global old_position;

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

Listing 4.1: File Matlab_PLC_connection.m [1]

The variables **t** and **old_position** are declared as global in order to allow the gestures system Matlab scripts to interact with the PLC connection and the last robotic arm position, respectively. In this phase the variables are initialized and at the end the connection is opened through the **fopen** function.

Establish the connection

After ensured that the computer is connected with the PLC and the ethernet interface address of the computer is set up to 192.168.0.241 (the address specified in the TCON component of the TIA Portal ladder diagram), it is possible to start the connection by executing the "Matlab_PLC_connection.m" Matlab script. In this way Matlab will wait for the PLC in order to create the connection object. At this point it is enough to push the "Connection Start" button on the PLC dashboard to connect the two devices and create the connection object **t**. Once this object is created, the Matlab server and the PLC can communicate.

4.2.1 Connecting Matlab to Python

The connection between the two softwares is accomplished using a Matlab server socket and a python client, respectively. The client is also the part of the project that is able to recognize the speech. In particular the Matlab server can be opened executing the script named "server.m":

```
srv = tcpip('127.0.0.1', 7777, 'NetworkRole', 'server','BytesAvailableFcnMode','
    terminator');
srv.BytesAvailableFcn = @connectionFcn;
fopen(srv);
global current_position;
current_position = [0; 0; 0; gripper.open];
```

Listing 4.2: File server.m [1]

The first parameter of the tcpip object will be the ip address of the client computer, the second parameter is the port of the socket to instantiate. The other parameters are needed to indicate the role of Matlab in the socket connection, in this case "server", and the mode to understand when there is a message available to be read in the socket buffer, in this case it looks for a CR/LF (the carriage return characters) to detect the end of the message. Another important parameter that is set is the **BytesAvailableFcn** (line 2), that is the callback that will be called every time there are bytes to be read in the input buffer of the socket. In this case the associated callback is the **connectionFcn** that is declared in the "connectionFcn.m" file:

```matlab
function connectionFcn(src, ~)
    global current_position;
    global old_position;
    global t;

    saved_positions = [];
    tmp_position = old_position;

    message = fscanf(src);
    message = strtrim(message);
    disp(message)

    if strcmp(message, "open") == 1
        tmp_position = move_robot(t, old_position,
            'rotation', current_position(1),
            'horizontal', current_position(2),
            'vertical', current_position(3),
            'gripper', gripper.open);
        if ~isequal(tmp_position, old_position)
            old_position = tmp_position;
            current_position(4) = gripper.open;
        end
    elseif strcmp(message, "close") == 1
        tmp_position = move_robot(t, old_position,
            'rotation', current_position(1),
            'horizontal', current_position(2),
            'vertical', current_position(3),
            'gripper', gripper.close);
        if ~isequal(tmp_position, old_position)
```

```
            old_position = tmp_position;
            current_position(4) = gripper.close;
        end
    elseif strcmp(message, "forward") == 1
        tmp_position = move_robot(t, old_position,
            'rotation', current_position(1),
            'horizontal', current_position(2) + 0.01,
            'vertical', current_position(3),
            'gripper', current_position(4));
        if ~isequal(tmp_position, old_position)
            old_position = tmp_position;
            current_position(2) = current_position(2) + 0.01;
        end
    elseif strcmp(message, "backward") == 1
        tmp_position = move_robot(t, old_position,
            'rotation', current_position(1),
            'horizontal', current_position(2) - 0.01,
            'vertical', current_position(3),
            'gripper', current_position(4));
        if ~isequal(tmp_position, old_position)
            old_position = tmp_position;
            current_position(2) = current_position(2) - 0.01;
        end
    elseif strcmp(message, "left") == 1
        tmp_position = move_robot(t, old_position,
            'rotation', current_position(1) - 10,
            'horizontal', current_position(2),
            'vertical', current_position(3),
            'gripper', current_position(4));
        if ~isequal(tmp_position, old_position)
            old_position = tmp_position;
            current_position(1) = current_position(1) - 10;
        end
    elseif strcmp(message, "right") == 1
        tmp_position = move_robot(t, old_position,
            'rotation', current_position(1) + 10,
            'horizontal', current_position(2),
            'vertical', current_position(3),
                'gripper', current_position(4));
            if ~isequal(tmp_position, old_position)
                old_position = tmp_position;
                current_position(1) = current_position(1) + 10;
            end
        elseif strcmp(message, "up") == 1
            tmp_position = move_robot(t, old_position,
                'rotation', current_position(1),
                'horizontal', current_position(2),
                'vertical', current_position(3) + 0.01,
                'gripper', current_position(4));
            if ~isequal(tmp_position, old_position)
                old_position = tmp_position;
                current_position(3) = current_position(3) + 0.01;
            end
        elseif strcmp(message, "down") == 1
            tmp_position = move_robot(t, old_position,
                'rotation', current_position(1),
                'horizontal', current_position(2),
                'vertical', current_position(3) - 0.01,
                'gripper', current_position(4));
            if ~isequal(tmp_position, old_position)
                old_position = tmp_position;
                current_position(3) = current_position(3) - 0.01;
            end
        elseif strcmp(message, "stop") == 1
            disp("The current position is: ");
            disp(current_position);
            saved_positions = [saved_positions; current_position];
            disp("Current saved positions: ");
            disp(saved_positions);
        else
            disp("Nothing");
        end

end
```

Listing 4.3: File connectionFcn.m [1]

This function is executed every time a new data is received from the client and accordingly

to the received message, it will perform a robot movement using the function **move_robot**. This last function is a high level Matlab api used to apply direct cinematic to the robotic arm through the array of radius and displacements of the joints of the robot. Each branch of the if statement corresponds to a robot movement, the resulting position of the robot is then saved inside the **tmp_position** variable, but it could happen that for a certain gesture, the robot has already reached the maximum displacement or rotation for a certain joint, thus the tmp_position will be equal to the **old_position** and there is no need to update the **current_position** variable, for this reason there is the if statement that checks the equality of tmp_position and old_position. One problem of this particular implementation is that it could bring to inconsistency between the robot position and the current_position variable (that contains the joints displacements, rotation and the gripper status, with respect to the initial position [0, 0, 0, 0]). In fact the displacement joints are moved by units of 1cm at a time and the rotational joint is moved by units of 10 degrees at a time, this could bring to situations in which, considering the "right" gesture, for instance less than 10 degrees are needed for the robot rotational joint to reach the limit switch, so the robotic arm moves anyway, the position changes and the first element of the current_position variable is increment by 10, but the robot moved by less degrees, so there is inconsistency. In order to resolve this problem it is possible to fine tune these displacement an rotation values, in order to trade off between movements execution speed and avoiding inconsistencies. Another way could be to derive the robot position from the old_position variable that contains a particular byte-based representation of the joint positions that requires 2 integer for each one of them, for reference on this part it could be useful to consult the documentation of the team of students [2] that developed the "move_robot" Matlab function or simply by inspecting the function.
Once the "server.m" is executed, Matlab will wait for the client to connect to the socket. At this point it is enough that the user runs from the client computer the "app.py" script:

```python
from connection.client import Client
...
if __name__ == "__main__":
    c = Client("192.168.43.78", 7777)
    c.connect()
    c.send("Hello server!")
```

```
...
```

Listing 4.4: File app.py - abstract of the connection part

That instantiate a client socket in the variable **c**, specifying the IP address of the Matlab server and the socket port. Than with the "connect" method, the connection will be established and with the "send" method a message is sent to the server. At this point on Matlab the variable **srv** will be instantiated, representing the socket object, and on the command line will appear the message sent from the python client. The app.py script uses the **Client** class of the file "client.py" contained in the subdirectory "connection" of the project:

```python
import socket


class Client:

    def __init__(self, host, port):
        self._HOST = host      # The server's hostname or IP address
        self._PORT = port      # The port used by the server
        self._s = None         # The TCP socket object

    def connect(self):
        self._s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self._s.connect((self._HOST, self._PORT))

    def disconnect(self):
        self._s.close()
        self._s = None

    def send(self, message):
        message += "\n"
        self._s.sendall(bytes(message, "ascii"))
```

Listing 4.5: File client.py[1]

4.3 Implementation of Speech Recognition

Before running the software, it is necessary to install the Speech_Recognition library, limited to 3.8+ Python; in order to do so it's necessary running the command:

`pip install SpeechRecognition`.

An internet connection is necessary for this step and, after the installation is over, the software is ready to run by calling the "import" command:

`import speech_recognition as sr`

```python
import speech_recognition as sr
from connection.client import Client

if __name__ == "__main__":

    # For Matlab
    global c
    c = Client("127.0.0.1", 7777)  # 172.22.32.1 192.168.6.177
    c.connect() # The software stops here in case it can't find a connection
    server
    c.send("Hello server!")

    # Command dictionary
commands = {
    "power": "shut down",
    "stop": "stop",
    "please go forward": "forward",
    "please go backwards": "backward",
    "please go up": "up",
    "please go down": "down",
    "please go left": "left",
    "please go right": "right",
    "please open": "open",
    "close": "close"
}

microphone_index = 1

# Speech recognition function
def recognize_speech():

    # Creating recognizer object
    r = sr.Recognizer()

    # Using microphone as audio source
    with sr.Microphone(device_index=microphone_index) as source:

        print("Speak now...")

        # Setting noise floor
        r.adjust_for_ambient_noise(source)

        # Recording audio from source
        audio = r.listen(source)

    try:

        # Writing down the audio as text
        text = r.recognize_google(audio)

        # Returning the command based on audio recorded
        return commands.get(text.lower(), "")

    except sr.UnknownValueError:

        print("I did not understand what you said.")

    except sr.RequestError as e:

        print(f"Error in speech recognition: {e}")

    return ""

def recognize_number():

    m = sr.Recognizer()

    with sr.Microphone(device_index=microphone_index) as source2:

        print("How many times?")
        m.adjust_for_ambient_noise(source2)
        audio2 = m.listen(source2)

    try:

        cycle = m.recognize_google(audio2)

        number = int(cycle)

        return number

    except sr.UnknownValueError:

        print("I did not understand the number.")

    except sr.RequestError as e:

        print(f"Error in speech recognition: {e}")

    return ""

# Main loop
while True:

    command = recognize_speech()

    if command == "shut down":

        print("Disabling speech recognition.")
        break

    elif command:

        message = command

        number_of_cycles = recognize_number()
        print(f"Command sent: {command} x{number_of_cycles}")
        if isinstance(number_of_cycles, int):
            while number_of_cycles > 0:
                c.send(message)
                number_of_cycles=number_of_cycles-1

    else:
        print("Awaiting command...")
```

Listing 4.6: File Speech_Recognition.py

The recognize_speech function works by recording the audio coming from the microphone, adjusting for ambient noise, transforming the audio into text and then the text into a command message, using the previously defined dictionary. The recognize_number function has the exact same structure and uses the same speech recognition, it's used for selecting the number of times the user wants the command to be performed.

CHAPTER 5

Conclusions and Future development

For accessing the project files it is possible to view the repository at: `https://github.com/LeleRuggeri/SpeechRecognitionFischertechnik`[1]

Bibliography

[1] Repository of the developed gesture detection system: fischertechnik-3d-robot-webcam-gestures-system. `https://github.com/davide-ciraolo/fischertechnik-3D-robot-webcam-gestures-system`. (Citato alle pagine 1, 8, 9, 11, 12, 13, 15, 16, 17, 20, 22 e 27)

[2] Repository of the fischertechnik-3d-robot developed by another team of students. `https://github.com/Giorgionocera/fischertechnik-3D-robot`. (Citato alle pagine 11 e 21)