Automated AI-based pick-and-place system

Enrico Catalfamo Vincenzo Aricò

November 13, 2023

Contents

Chapter 1

Introduction

1.1 Premises and goals

The project aims, as the title suggests, to create an automated pick-and-place system through the use of Machine Learning techniques. The basic elements involved in the implementation of the project are the following:

- **A robotic arm**, which is the main player in pick-and-place operations;
- **A conveyor belt**, which conveys the objects to the pre-established point where they are picked up by the robotic arm;
- **A PLC (Programmable Logic Controller)**, responsible for direct control of the robotic arm;
- **A server machine**, which interfaces with the PLC via an Ethernet port to control the robotic arm and on which runs the machine learning model (a convolutional neural network) responsible for object recognition;
- **A client machine**, which connects to the server machine to start or stop the system using a software with a graphical interface, which also

allows you to visually monitor the working environment of the robotic arm.

A more accurate description of the hardware involved (as well as a more exhaustive list) will be made in the subsequent chapters. The basic idea is to have a "proxy" (what we have defined as "server machine"), on which all those operations that generate a certain workload are carried out (management of the webcam flows necessary for monitoring and prediction on objects, loading and starting the machine learning model, management of the robotic arm via commands sent on a socket to the PLC), to which a client can connect, to manage and monitor the system, via an ad-hoc software.

1.2 High level system architecture

The diagram representing the architecture of the system at a high level of abstraction is the following:

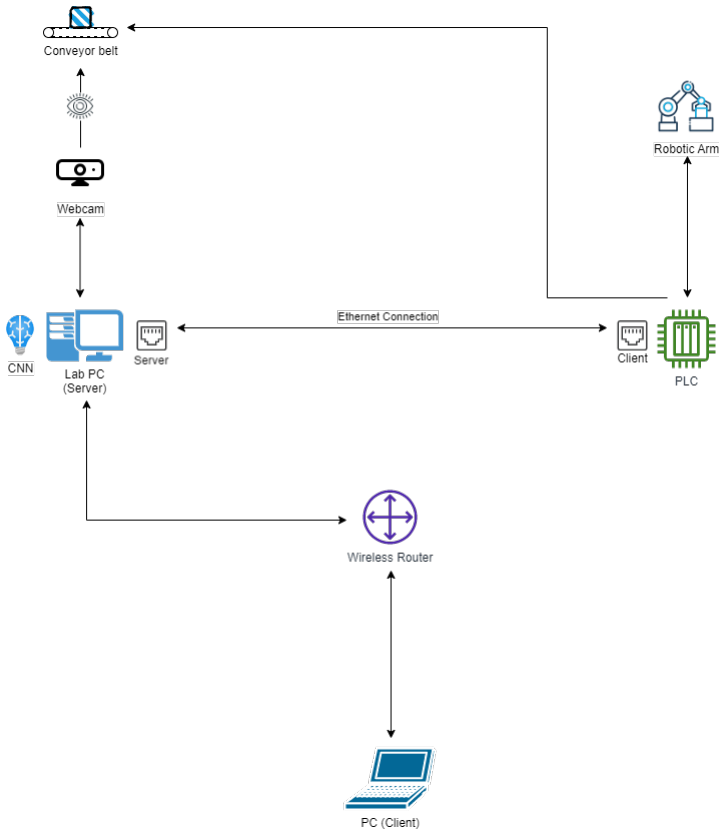


Figure 1.1: High level system architecture

Clearly, the conveyor belt is managed by the PLC through the use of a photoelectric sensor which allows determining the presence or absence of an object to be picked: when an object is detected, the PLC ceases to power the conveyor belt, which stops, allowing the robotic arm to carry out all the necessary operations. More specifically, the logical workflow is the following:

1. The server is started and connects to the PLC on two different sockets: the first one is useful for sending commands to the PLC for managing the robotic arm, while the second one is used by the PLC to send a packet to the server whenever an object is detected, in order to start the server-side AI classification algorithm. In the meantime, the server remains listening on two other sockets (one for exchanging messages

aimed at communication, the other for sending the monitoring webcam stream), waiting for a client;

2. The client connects to the server via the client software GUI, receiving the monitoring webcam stream. By clicking on the start button, the client software sends a **start** message to the server, which starts the system;

3. The server waits for an interrupt from the PLC. When it arrives, signaling the presence of an object, a prediction webcam is used to obtain a frame of the object on which to make a prediction using the AI model;

4. Depending on the prediction, the robot is commanded to pick up and place the object in one position rather than another;

5. Once the robot has finished this pick-and-place cycle, it starts again from point 3, and continues like this until the client disconnects or decides to stop the system.

While the client could be located anywhere, the server machine must be located in a place near the robotic arm as the connection between the PLC and the server machine is made through an ethernet cable.

Chapter 2

Technologies Involved

In this chapter, all the technologies involved in the project will be covered, both hardware and software. We will first list and describe the hardware components used; subsequently, we will review all the software, programming languages and libraries that have come to our aid. It is important to underline that the following is not just a mere list: together with the theoretical notions inherent to the various technologies, certain project choices will be explained and motivated.

2.1 Hardware

Although the heart of the project is the software part, it is of great importance to describe all those hardware components that were used and without which it would not have been possible to implement it. In the following section, you will find an explanatory list of these components, together, where necessary, with the reasons behind their choice.

2.1.1 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

2.1.2 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.

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

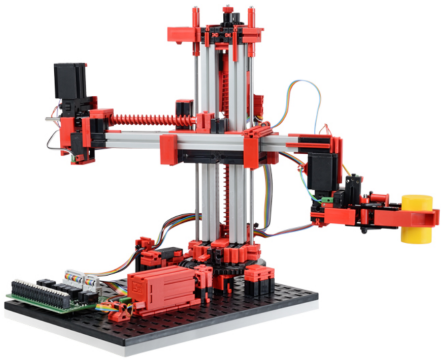


Figure 2.2: Fischertechnik 3D-Robot 24V

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.

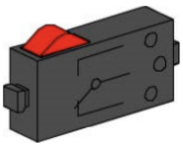


Figure 2.3: Mini switch

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.

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.

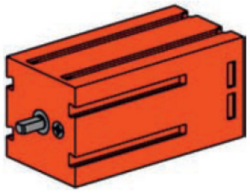


Figure 2.4: Motor encoder

2.1.3 Conveyor Belt

The conveyor belt was originally part of an old project and was retrieved at the university, then repurposed for our aims. Almost all the parts that generated unnecessary bulk were removed from the wooden support, as well as all the electronic components that could not have a role in our project. What is left is visible in figure 2.5:

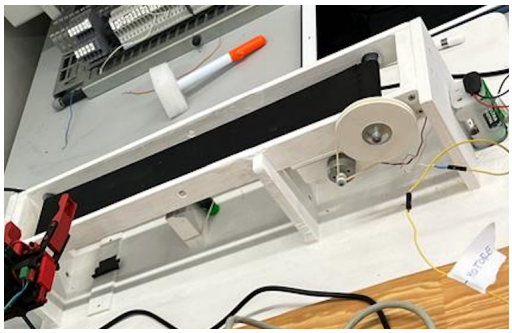


Figure 2.5: Conveyor Belt

To make it usable, it was subsequently necessary to re-tension the belt and replace the "transmission" (a common elastic band). Once done, we took care of checking the voltage necessary to power the DC motor and allow the belt to move at an acceptable speed. After some tests, we came to the conclusion that the optimal voltage for the motor at our disposal was 3V.

2.1.4 Photoelectric sensor E3F-DS10C4

Figure 2.6: E3F-DS10C4 sensor

The E3F-DS10C4 sensor (Figure 2.6) is a cylindrical diffuse reflection photoelectric sensor with integrated circuit technology, which works as a normally closed switch (normally outputs a high value and switches to a low value when in which it detects the presence of an object). The choice fell on this sensor for the following reasons:

- **Operating technology**: diffuse reflection. Unlike some inductive and capacitive proximity sensors, photoelectric sensors can detect the presence of objects made of any material (as long as they reflect light), and are therefore more versatile;
- **Supply voltage**: this sensor can be powered with a variable voltage between 10VDC and 30VDC, perfectly in line with the PLC power output (24VDC);

- **Low cost**: the purchase price is relatively low (about 10 euros), especially if you take into account the elasticity, detection speed and robustness of the sensor;
- **Performance**: the sensor is capable of detecting an object placed at 10 cm (adjustable) away by switching in very short times (2.5 ms max);
- **Mounting support**: The geometry of the sensor, together with the included fixing washers, allows the sensor to be mounted and fixed in an optimal way, guaranteeing precision in detections.

Both the output and the power cable of the sensor were connected to the PLC. Through the ladder diagram, the PLC monitors the status of the relevant input (sensor output) and subsequently carries out all the necessary operations (stop conveyor belt and send packet to the server machine to perform AI prediction of the type of object).

2.1.5 24V Relay

Figure 2.7: The 24V relay used for the project

A 24VDC relay (Figure 2.7) is an electromechanical device used to control low voltage electrical circuits, operating as a voltage-controlled switch. The operation is very simple: through a 24VDC supply voltage, a solenoid coil is activated, which generates a magnetic field. The interaction between the

magnetic field generated by the coil and a mechanical moving contact allows a separate electrical circuit to be opened or closed, allowing the passage of current or interrupting it.
In our case, the relay was indispensable for controlling the conveyor belt via PLC: direct control, in fact, would not have been possible due to the high output voltages of the PLC; in this way, however, it is possible to connect the PLC output to the relay coil, switching between the 3V and 0V output to power the motor. Actually, it would have been possible to connect the sensor output directly to the relay to control the conveyor belt motor. This, however, would have caused the following problems:

1. **Temporization**: The conveyor belt would restart as soon as the sensor no longer detected the presence of an object. This could have caused collisions between the robot and objects arriving at the picking point and moving on the conveyor belt. By connecting the relay to the PLC output, however, it is possible to insert a timer in the ladder diagram to delay the restart of the conveyor belt.

2. **Use of resources**: The server machine would have had no way of being informed about the presence or absence of an object. Consequently, it would have been necessary to take frames from the webcam at a programmed interval to verify the presence of an object and carry out the recognition via AI, using the CPU unnecessarily.

2.1.6 USB Webcams

As we anticipated in the introduction chapter, in order to realize the project it was necessary to use two webcams: one that took care of environmental monitoring and the other, fixed on the conveyor belt, for object recognition. Since there are no particular requirements for choosing a webcam, except a decent resolution (which is now a common feature of all the devices of this kind on the market), we will not delve into the matter further.

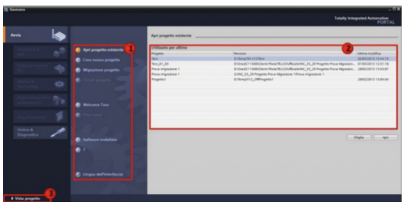
2.2 Software

In this chapter we will discuss from a theoretical point of view all the softwares and programming languages used, with the related libraries.

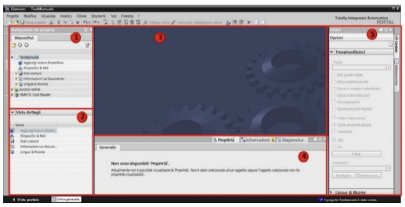
2.2.1 TIA Portal

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

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.

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 Python

Python is an object-oriented programming language suitable for application development, scripting, numerical computing and system testing. It is one of the most used languages in the scientific field due to the simplicity

of its syntax and the large availability of libraries oriented to experimentation, data analysis and the construction of graphs. In particular, the Numpy and PyPlot libraries are among the most used in this context. Furthermore, with the growing development of artificial intelligence algorithms, many important libraries have also been created such as PyTorch, Keras, Tensorflow and OpenCV, which allow to apply different types of Machine Learning algorithms, not only on numerical or textual data but also on audios, images and videos.
Python is the language chosen for development of this project because is simple and has very powerful libraries oriented to computer vision, that is the technique able to recognize objects inside images, like the webcam frames.

2.2.2.1 OpenCV

OpenCV is an open-source library widely used for image processing and computer vision. It is designed to provide a high-level toolset for developers and researchers to create applications involving real-time image processing. Although it is primarily written in C++, it offers interfaces for Python, Java, and MATLAB, providing flexibility and adapting to a wide range of projects. The library contains over 2500 optimized algorithms, covering a wide range of areas such as face detection, object recognition, motion tracking, camera calibration, etc. .
Its versatility makes OpenCV a popular choice for a wide range of applications. In this context, it was mainly used to interface the server machine with the connected webcams, contributing to the management of the video streaming between the server and the client machine.

2.2.2.2 Zmq

ZMQ is a high-speed, low-latency messaging library designed to simplify the development of distributed, scalable applications. It is open-source and

offers an asynchronous and stateless communication architecture, which allows the exchange of messages between system components efficiently.
The library is primarily written in C, but also offers bindings for many other programming languages, including Python. This makes ZMQ accessible to a wide range of developers and allows integration with a variety of technology stacks.
The ZMQ library provides a variety of communication models, including the publish/subscribe model, abstracting lower-level complexity and providing useful, powerful, and easy-to-integrate functionality within applications. In particular, in the context of this project, ZMQ was used to create a socket with a pub/sub model, useful for managing MJPEG (Motion JPEG) video streaming from the server to the client.

2.2.2.3 Pillow

Pillow is an open-source library for image processing in Python. It is a Python Imaging Library (PIL) project fork and offers a wide range of functionalities for image manipulation, editing, and generation.
Pillow provides a simple and intuitive interface for working with images in different formats, including JPEG, PNG, TIFF, BMP and many others. Images can be opened, saved and converted between different formats, allowing developers to easily manage images with different format requirements. In the context of this project, it was used for the management of received frames on the client side within the graphical interface, created with Tkinter.

2.2.2.4 Tensorflow

TensorFlow is an open-source artificial intelligence library developed by Google. It is designed to facilitate the creation and training of machine learning models, especially neural networks, on a wide range of devices and platforms.

TensorFlow provides a wide range of tools and features for developing AI models. Its main architecture is based on a data flow called a "computational graph", where nodes represent mathematical operations and links between nodes represent data flowing through the system.
The library supports CPU and GPU computing, allowing developers to leverage the computing power available across devices. Additionally, TensorFlow offers support for distributed computing, allowing you to distribute computation across multiple machines to tackle larger problems and improve performance.
TensorFlow offers a wide range of machine learning algorithms and pre-built neural networks, such as convolutional neural networks (CNN) and recurrent neural networks (RNN). These algorithms can be used for a variety of tasks, such as, in this case, image classification. By providing pre-trained neural networks on large datasets, Tensorflow allows you to carry out Transfer Learning on tested and functioning models, achieving very high performance. In our case, Tensorflow was used to carry out Transfer Learning on the VGG16 network, training only the Dense layers we inserted.

2.2.2.5 NumPy

NumPy is an open-source library for scientific computing in Python. It is widely used to perform efficient operations on multidimensional arrays and provides a wide range of capabilities for scientific data processing.
NumPy offers a multidimensional array object called ndarray, which allows you to store and manipulate data efficiently. Ndarrays enable efficient operations on large amounts of data, such as vector calculations, element-wise mathematical operations, and broadcasting.
The library also offers advanced mathematical functions to perform linear algebra, Fourier transforms, statistics, and random number generation operations. These features allow developers to perform complex calculations and

data analysis efficiently and accurately.
In our case, NumPy was mainly used to convert (manually, as well as with reference to data structures) the Matlab code [7] interfacing the PLC and the robotic arm into Python code, avoiding the use of interface libraries between Matlab and Python, helping to make the application robust and monolithic and improving code consistency. Furthermore, the NumPy library was used, together with the other libraries, to manage the images coming from the webcams.

2.2.2.6 Yaml

YAML (YAML Ain't Markup Language) is a human-readable data serialization format that is easily parsable by programming languages. It is often used for configuring applications, exchanging structured data, and representing documents. In our project, it was used to create a configuration file that includes all the variable parameters depending on the specific situation (server and PLC IP addresses, interfacing ports, webcam IDs, etc.), in such a way as to allow you to configure the system quickly and, above all, without having to directly modify the code.

Chapter 3

Implementation

Below, we will cover the implementation part of the project, as well as the ways in which the system was actually created. We will first discuss the details of the hardware implementation (all the various circuit connections between the devices that make up the system), then we will move on to describe the features of the code that allow all the components to carry out their work in an effective and coordinated manner.

3.1 Hardware

3.1.1 Circuit model

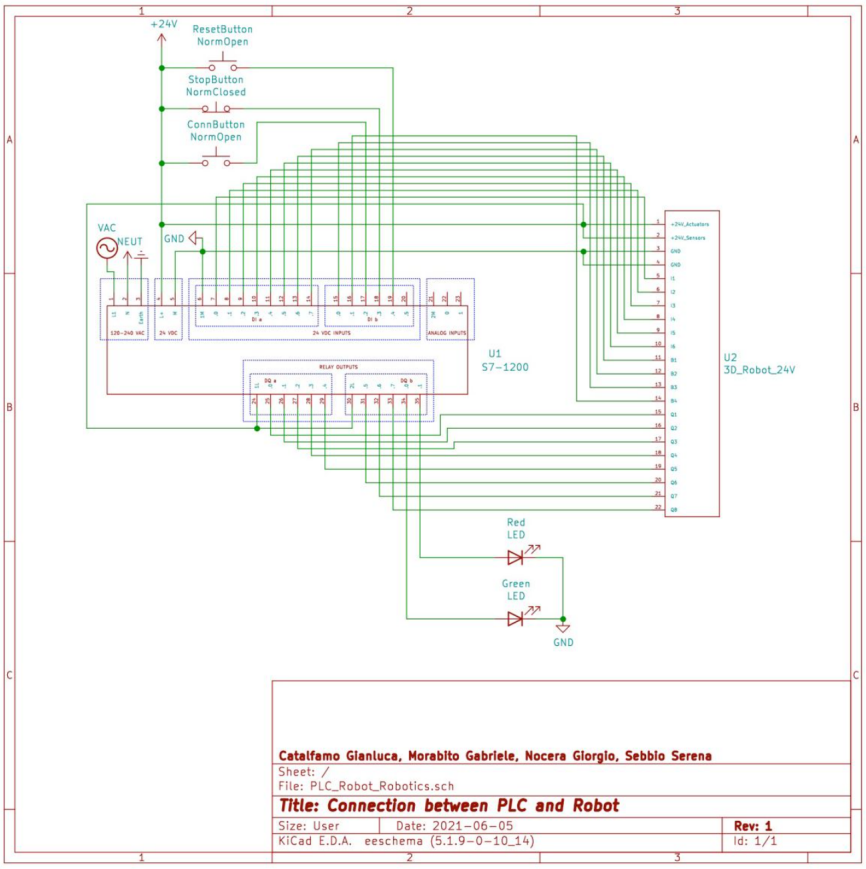


Figure 3.1: Initial schematic

The circuit in Figure 3.1 represents the wiring that was found in the laboratory, physically realized by other students [3]. We started from this circuit to add the hardware related to our system in a "transparent" way (trying not to compromise the circuit already present).

3.1.1.1 Photoelectric sensor E3F-DS10C4 connections

The E3F-DS10C4 photoelectric sensor has three wires that need to be connected to the PLC:

- **power supply (brown)**: connected to pin L+ (24 VDC port) of the PLC;
- **ground (blue)**: connected to pin M (24 VDC port) of the PLC;
- **output signal (black)**: connected to the last available input pin, i.e. pin DIb5 (24 VDC INPUTS port) of the PLC.

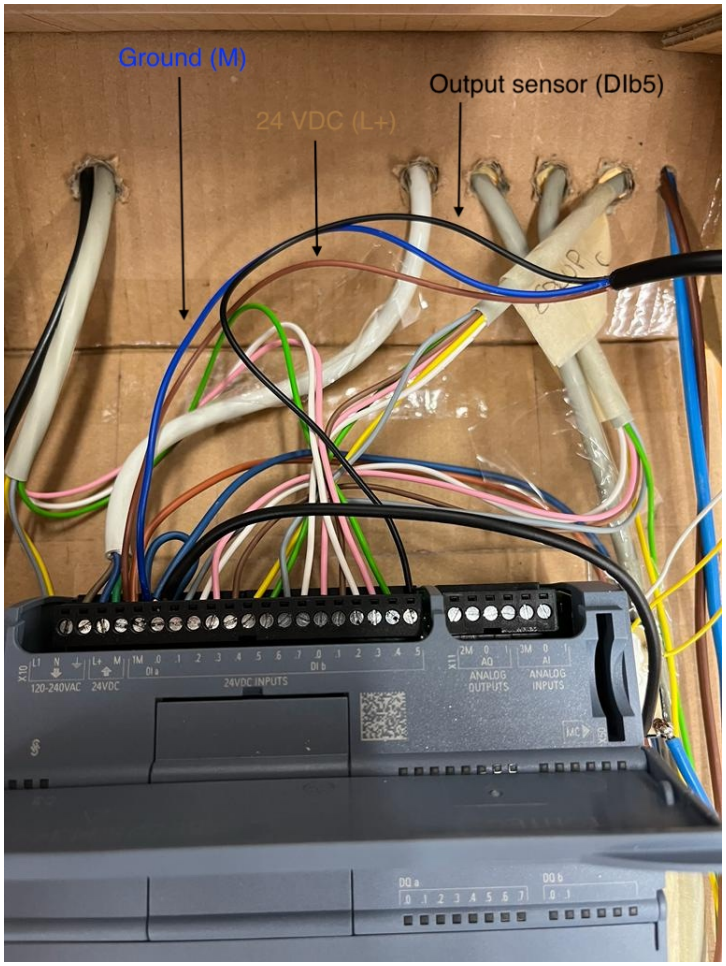


Figure 3.2: E3F-DS10C4 connections

3.1.1.2 Conveyor Belt connections

The DC motor, capable putting in motion the conveyor belt, must be powered with 3 VDC (2 1.5 VDC batteries). The power supply to the motor must be immediately interrupted when an object is detected by the photoelectric sensor, while it must be restored after a few seconds when the detected object is picked up by the manipulator, in order to avoid collisions between the end-effector and the objects arriving at the pick position. Power to the motor is provided by a relay controlled by voltage from a 24 VDC output pin of the PLC. Since there were no output pins available, the red LED connected to pin DQb1 (RELAY OUTPUTS port) was disconnected in order to use the pin to control the relay input voltage. This is how it works:

- **pin DQb1 = 0 VDC**: object not detected, therefore the conveyor belt can be powered;
- **pin DQb1 = 24 VDC**: object detected, therefore power to the conveyor belt must be immediately cut off.

To supply or interrupt power to the conveyor belt, two separate circuits are necessary, as in Figure 3.3. The circuit on the left (with the battery) will power the motor and close ("activate") when **pin DQb1 = 0 VDC**, while the circuit on the right (without the battery) will cut power to the motor and close ("activate") when **pin DQb1 = 24 VDC**. Obviously, when one of the two circuits is closed, the other one will be open.

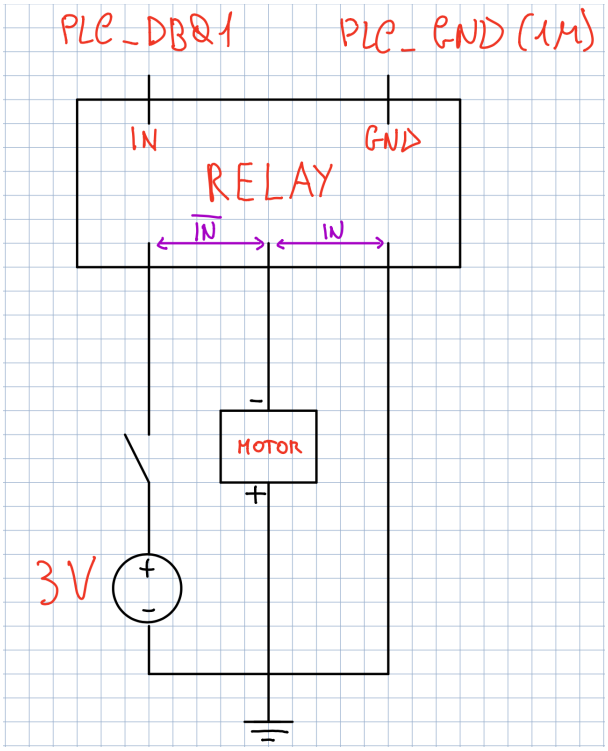


Figure 3.3: Relay schematic

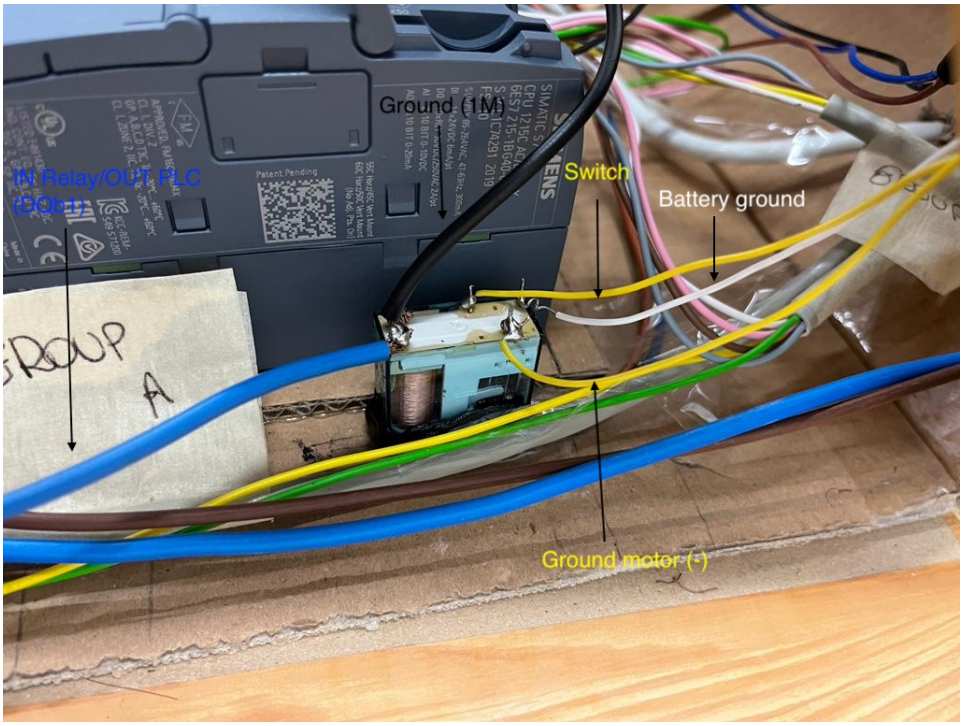


Figure 3.4: Relay connections

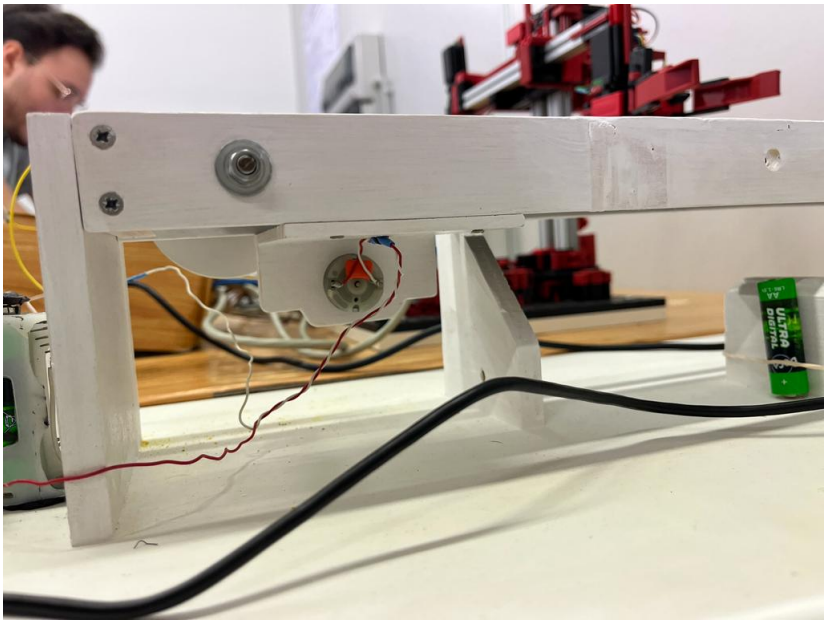


Figure 3.5: Motor position on conveyor belt

Figure 3.6: Labeled relay connections to separate circuits

3.1.2 Scene image

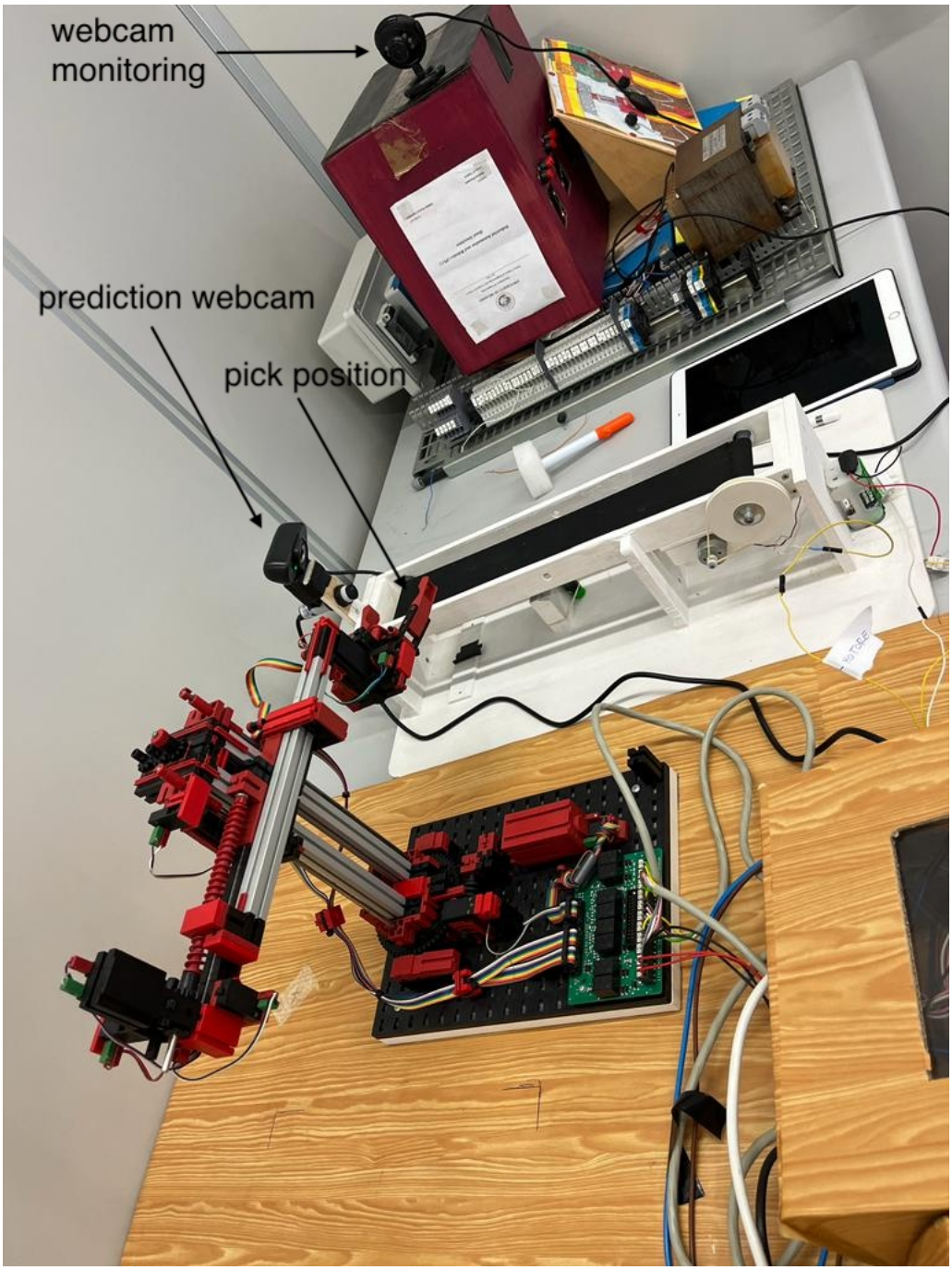


Figure 3.7: Scene image

3.2 Software

The software is mainly composed by the following elements:

- The Communication module, which defines a JSON-based application-level communication protocol to allow the exchange of system control messages between client and server;
- The ServerClass class, which is responsible for orchestrating all the functions related to environment monitoring, robot movement and object recognition;
- A convolutional neural network, which is called at the appropriate time to perform object recognition;
- A script for the client, which generates a graphical interface through which it is possible to connect to the server, start/stop the system and monitor its behavior;
- A configuration file, containing all the parameters relating to the connections between client, server and PLC, which allows the system to be reconfigured easily and quickly depending on the context.

Server-related tasks are conveniently handled by separate threads, creating a multi-threaded application.

3.2.1 AI Model

To simplify and abstract the object recognition part, foam rubber cubes were created on which a geometric shape between a square and a triangle was drawn. Recognizing the object, therefore, boils down to recognizing the geometric shape present on the cube. The point at which the cube is released once grabbed by the robot depends on the latter.

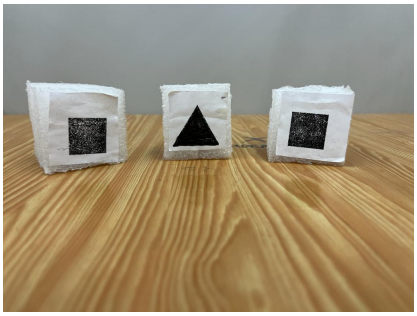


Figure 3.8: Cubes

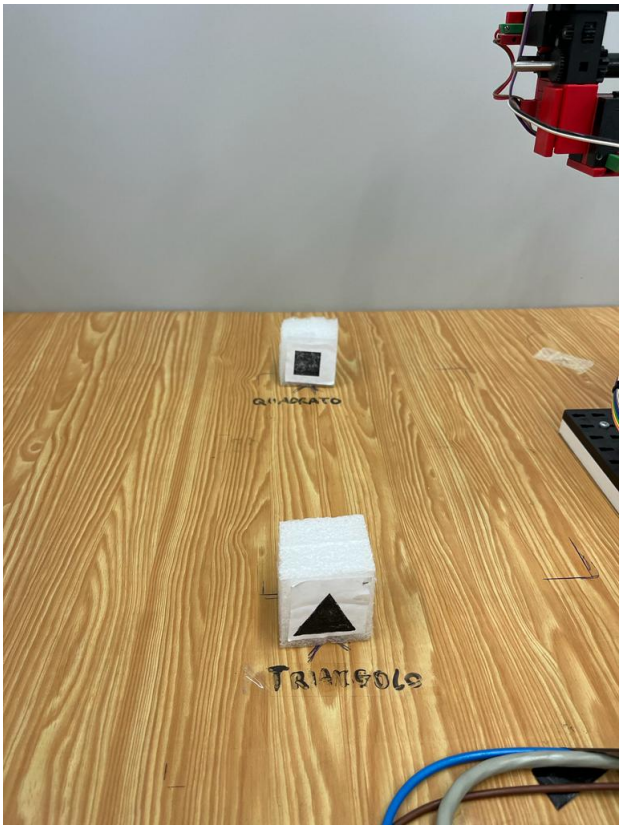


Figure 3.9: Targets

The model used for shape recognition is a convolutional neural network, in particular the VGG16 network, which has been modified in its final Dense layers to adapt it to the task of our interest. The network is able to distin-

guish between the two geometric shapes, as well as verify that a geometric shape is actually present: if this does not happen, the system must notify the event. This implies that the network must be trained in such a way to be able to detect the absence of geometric shapes.

3.2.1.1 Creating the dataset

We could have created the dataset "by hand", taking pictures of the previously crafted cubes and feeding them to the network during the learning phase. However, this approach would inevitably have involved the use of a small dataset, leading to overfitting and, therefore, poor performance in the inference phase (the network would not have been able to generalize correctly). The path we preferred to follow, however, is that of using a synthetic dataset.

This dataset was obtained using an image generator [6] containing geometric shapes (randomly rotated and scaled) of different colors on a homogeneous background. The software, however, creates images containing not only triangles and squares, but also other geometric shapes. Since the generation of images takes a relatively long time, especially when the number of examples to be generated is quite high, it was necessary to make some slight modifications to the code to ensure that only the geometric shapes of interest to us were generated. Furthermore, to make the network able to recognize the absence of geometric shapes, we ensured that images were generated without geometric shapes in the background. With these methods, a balanced dataset of 15000 images (5000 per class) was generated.

a) Triangle **b)** Square **c)** Absence

Figure 3.10: One image per class

3.2.1.2 Training the model

The model consists of an input layer with shape (200, 200, 3), a rescaling layer to bring the pixels into a range between 0 and 1, the pretrained hidden layers of the VGG16 network and three final dense layers to train (one with 32 neurons, one with 16 and the last one, of course, with 3 neurons and softmax activation function). The Adam optimizer was used, with a starting learning rate of 0.0001. The reference metric is accuracy. The network was trained for a single epoch, to avoid overfitting. The ratio between the cardinality of the training dataset and that of the test dataset is 80-20: 12000 samples to train the network and 3000 to test its generalization capabilities. The model achieved an accuracy of 94% on the training set and 97.5% on the test set.

3.2.2 The Communication module

The Communication module allows client and server to exchange messages about system control in a simple way. Inside, two functions are defined: receive() and send(). Both functions take advantage of the tools made available by Python sockets and the JSON module. The send() function, very simply, creates a dict using the input parameters, transforms it into JSON, serializes it and sends it to the socket.

```
def send(socket, type: str, payload: str):

    packet = {
        "type" : type,
        "payload" : payload
    }

    json_packet = bytes(json.dumps(packet), 'utf-8')
    socket.send(json_packet)
    return packet
```

Figure 3.11: The send() function

The receive() function listens on the socket (collecting packets) until it has received the entire JSON. This is verified, thanks to the fact that JSON

is highly structured, by counting the curly braces contained in the received packets. When the counter reaches zero (i.e. when as many opening braces as closing braces have been received), the JSON is returned in full (as a dict) to the caller.

```
def receive(socket):
    message = bytes()
    n = 0

    while True:
        try:
            content = socket.recv(1024)
        except ConnectionResetError:
            return None
        if not content:
            return None
        message+=content

        for char in str(content, 'utf-8'):
            if char == '{':
                n+=1
            if char == '}':
                n-=1

        if n == 0:
            break

    message = json.loads(message)
    return message
```

Figure 3.12: The receive() function

3.2.3 Client-side code

The client-side code is very simple. Through Tkinter, a graphical interface is created, with the aim of giving the user a simple and intuitive means for managing the system. The script takes the server connection parameters from the configuration file and defines some main elements:

- A "Connect" button, which allows you to connect to the server, receiving the video stream and receiving/sending messages related to system control;
- A "Disconnect" button, which closes both sockets (both the one related

to the video stream and the one related to the exchange of control messages);

- A "Start" button, which starts the system (sending a control message to the server), allowing the robot to start sorting the cubes;
- A "Stop" button, which stops the system;
- A window showing the video stream of the working environment;
- A box in which any messages returned by the server are shown.

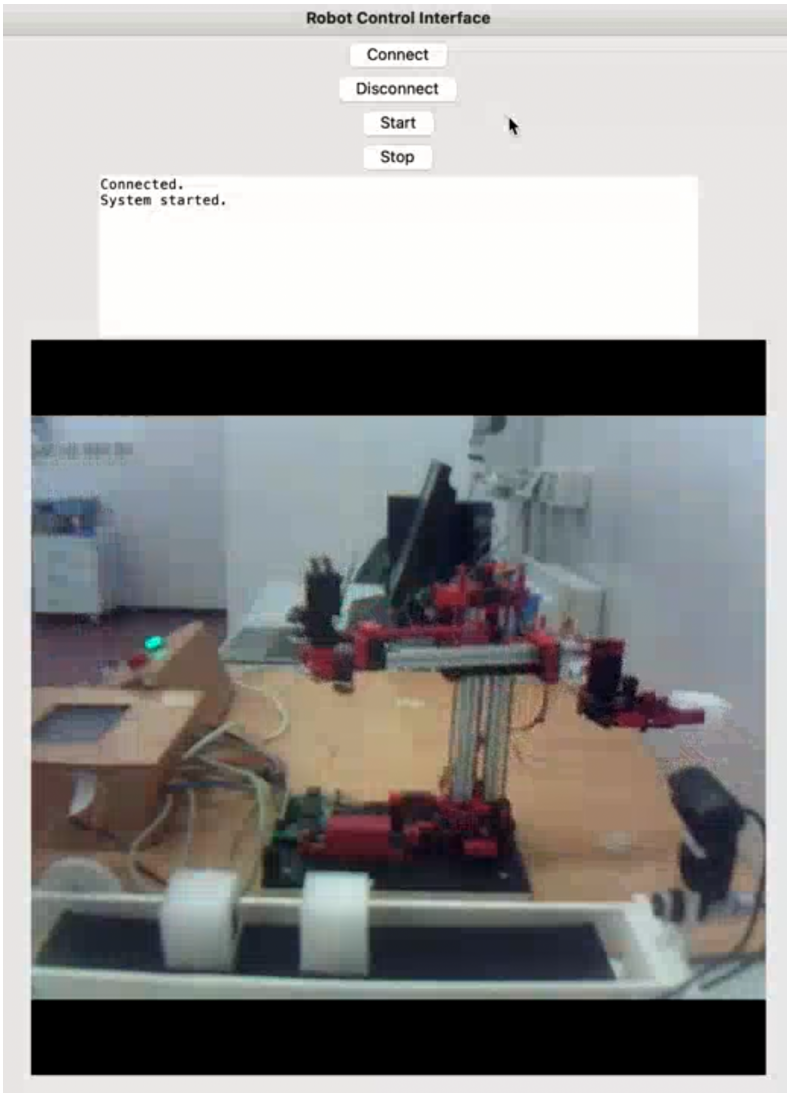


Figure 3.13: Client control interface (GUI)

All these elements (in particular the video streaming window and the message box) are updated every 4 milliseconds, following the logic defined in the update_root() function.

```python
def update_root():
    frame = footage_socket.recv_string()
    img = base64.b64decode(frame)
    npimg = np.frombuffer(img, dtype=np.uint8)
    source = cv2.imdecode(npimg, cv2.IMREAD_COLOR)
    img = cv2.cvtColor(source, cv2.COLOR_BGR2RGB)
    #img = img.resize((600,600))
    img_pil = Image.fromarray(img)
    img_tk = ImageTk.PhotoImage(img_pil)
    video_label.config(image=img_tk)
    video_label.img = img_tk

    if len(msg_list) != 0:
        msg = msg_list.pop(0)
        text_box.config(state = 'normal')
        text_box.insert('end', msg)
        text_box.config(state = 'disabled')
    if connected == 1:
        root.after(4, update_root)
    else:
        footage_socket.disconnect(f'tcp://{host}:{video_port}')
        video_label.img = None
```

Figure 3.14: The update_root() function

The function takes a frame from the video stream socket, shows it on the screen (replacing the previous one) and checks that there are elements in the control message queue, in which case it shows them in the related box. The GUI update, which occurs every 4 milliseconds, is carried out only if the client is connected to the server (otherwise, it is not necessary to update the screen). The other functions defined in the script, however, simply deal with connecting or disconnecting the client from the server, updating the state variables accordingly, or sending specific control messages to the server via the Communication module (start and stop, specifically).

```python
def connect():
    global context
    global footage_socket
    global comm_socket
    global connected

    if connected == 0:
        footage_socket = context.socket(zmq.SUB)
        comm_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        footage_socket.connect(f'tcp://{host}:{video_port}')
        footage_socket.setsockopt_string(zmq.SUBSCRIBE, b''.decode('utf-8'))
        comm_socket.connect((host, comm_port))
        connected = 1
        comm_thread = threading.Thread(target=socket_receiver, args=(comm_socket, msg_list))
        comm_thread.start()
        text_box.config(state = 'normal')
        text_box.insert('end', "Connected.\n")
        text_box.config(state = 'disabled')
        update_root()
```

Figure 3.15: The connect() function

```python
def disconnect():
    global connected
    global comm_socket

    if connected == 1:
        comm_socket.close()
        connected = 0
        text_box.config(state = 'normal')
        text_box.insert('end', "Disconnected.\n")
        text_box.config(state = 'disabled')
```

Figure 3.16: The disconnect() function

3.2.4 Server-side code

The server-side script imports the ServerClass class, takes the configuration parameters from the configuration file, creates a ServerClass instance and calls the start method to instantiate the server.
The ServerClass class defines two methods: the initialization method (__init__), and the start method, which instantiates and runs the server. The input parameters of the __init__ function must be contained in a dictionary, and are:

- The IP address of the network interface on which the server to which the client must connect will be instantiated, as well as the related ports;
- The IP address of the network interface on which the server to which the PLC must connect will be instantiated, as well as the related ports.

Subsequently, in the same function, all the objects necessary for the correct functioning of the server are created, namely:

- Two cv2.VideoCapture objects, which provide a handle for the two webcams connected to the system (object recognition and monitoring of the working environment);
- The sockets to be bound to the IPs and ports above;
- A variable containing, as a string, the current state of the system (initially "STOP");
- A numpy array containing the initial position of the robotic arm (initialized to zero);
- Two Thread objects for separate management of the robot's movement and video streaming;
- A threading.Event object to ensure correct synchronization between system operation and commands sent by the client.

```python
class ServerClass():

    def __init__(self, params: dict):
        self.host_pc = params["host_pc"]
        self.comm_port = params["comm_port"]
        self.video_port = params["video_port"]
        self.host_plc = params["host_plc"]
        self.plc_port_1 = params["plc_port_1"]
        self.plc_port_2 = params["plc_port_2"]
        self._footage_camera = cv2.VideoCapture(params["footage_cam_ID"])
        self._AI_camera = cv2.VideoCapture(params["AI_cam_ID"])
        self._AI_model = load_model(params["AI_model_path"])
        self._video_socket = zmq.Context().socket(zmq.PUB)
        self._comm_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self._plc_socket_1 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self._plc_socket_2 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self._client_sync = threading.Event()
        self._system_state = "STOP"
        self._initial_position = np.array([[0],[0],[0],[0],[0],[0],[0],[0]])
        self._video_thread = threading.Thread(target=camera_handle, args=(self._video_socket,
                                                                          self._footage_camera))
        self._robot_thread = threading.Thread(target=robot_control, args=(self._plc_socket_1,
                                                                          self._plc_socket_2,
                                                                          self._initial_position,
                                                                          self._client_sync,
                                                                          self._AI_camera,
                                                                          self._AI_model))
```

Figure 3.17: ServerClass __init__ function

The start() function binds the server sockets to the addresses and ports defined in the previous function and starts the two threads related to the robot movement and the video stream, then listens on the communication socket with the client, waiting for a connection. Once the client is connected, the server waits to receive messages from it, as a result of which it updates its state and interacts with the robot's movement thread via the threading.Event object, stopping or starting the sorting of the cubes.

```python
def start(self):
    self._video_socket.bind(f'tcp://{self.host_pc}:{self.video_port}')
    self._comm_socket.bind((self.host_pc, self.comm_port))
    self._plc_socket_1.bind((self.host_plc, self.plc_port_1))
    self._plc_socket_2.bind((self.host_plc, self.plc_port_2))
    self._video_thread.start()
    self._comm_socket.listen()
    self._plc_socket_1.listen()
    self._plc_socket_2.listen()
    self._robot_thread.start()
    print("Server listening.")
    while True:
        conn, addr = self._comm_socket.accept()
        print(f"Client connected with address and port {addr}.")
        while True:
            message = receive(conn)
            if message is None:
                if self._system_state == "RUNNING":
                    self._client_sync.clear()
                    self._system_state = "STOP"
                    print("Due to client disconnection, the system has been stopped.")
                print("Client disconnected.\nServer listening.")
                break

            if message["payload"] == "START":
                if self._system_state == "STOP":
                    self._system_state = "RUNNING"
                    self._client_sync.set()
                    send(conn, "message", "System started.\n")
                else:
                    print("A Start command has been received, but the system is already running.")
                    send(conn, "message", "A Start command has been received, but the system is already running.\n")


            if message["payload"] == "STOP":
                if self._system_state == "RUNNING":
                    self._client_sync.clear()
                    self._system_state = "STOP"
                    print("System stopped.")
                    send(conn, "message", "System stopped.\n")
                else:
                    print("A Stop command has been received, but the system is not running.")
                    send(conn, "message", "A Stop command has been received, but the system is not running.\n")
```

Figure 3.18: ServerClass start() function

The camera_handle() function, linked to a separate thread, does nothing more than taking a frame from the webcam, encoding it first in JPEG, then in base64 and finally sending it over the socket, ready to be picked up by the client. This is done every 0.05 seconds.

```python
def camera_handle(socket, camera) -> None:

    while True:
        _, frame = camera.read()
        _, buffer = cv2.imencode('.jpg', frame)
        jpg_as_text = base64.b64encode(buffer)
        socket.send(jpg_as_text)
        sleep(0.05)
```

Figure 3.19: The camera_handle() thread function

The robot_control function, tied to the robot control thread, works as follows:

- When the thread is started, it listens on the socket, waiting for the PLC to connect;
- Once the PLC is connected, it verifies that the system is in running state, checking that the event object is set;
- If the system is running, it waits to receive notification from the PLC about the presence of an object in the pick position;
- Once it receives this message, it takes a frame from the object recognition webcam, withdraws the part of the image containing the geometric shape and feeds it to the ML model;
- Depending on the prediction, the robot is commanded to follow one path rather than another. If the predicted class is the third, corresponding to the absence of geometric shapes, this is notified to the server application, inviting the operator to remove the object from the conveyor belt.

The function that actually carries out the movement of the robotic arm, move_robot, is nothing but the Python version of the Matlab function of the same name defined in the project which can be consulted at the link [7] in the bibliography.

```python
def robot_control(socket_1: socket.socket,
                  socket_2: socket.socket,
                  pos: np.array,
                  event: Event,
                  cam: cv2.VideoCapture,
                  model) -> None:
    while True:
        robot_conn, _ = socket_1.accept()
        print("Robot motion control connected.")
        conveyor_conn, _ = socket_2.accept()
        print("Conveyor belt control connected.")
        print("PLC connected. System started.")
        n = 0
        while True:
            while event.is_set():
                msg = conveyor_conn.recv(1024)
                if "START" in str(msg):
                    print("Object detected by the photosensor.")
                    _,frame = cam.read()
                    frame = frame[450:750, 800:1200]
                    cv2.imwrite('model/predictions/' + str(n) + '.jpg', frame)
                    n += 1
                    frame = cv2.resize(frame, (200, 200))
                    frame = convert_to_tensor(frame)
                    frame = expand_dims(frame, axis=0)

                    prediction = model.predict(frame)
                    print(prediction)
                    class_detected = np.argmax(prediction)

                    if class_detected == 0:
                        pos = move_robot(robot_conn, pos, rotation = -3.5, horizontal = 20, vertical = -0.03, gripper = 0)
                        pos = move_robot(robot_conn, pos, rotation = -3.5, horizontal = 20, vertical = -0.03, gripper = 1)
                        pos = move_robot(robot_conn, pos, rotation = 0, horizontal = 0, vertical = 0, gripper = 1)
                        pos = move_robot(robot_conn, pos, rotation = 160, horizontal = 0, vertical = 0, gripper = 1)
                        pos = move_robot(robot_conn, pos, rotation = 160, horizontal = 20, vertical = -0.13, gripper = 1)
                        pos = move_robot(robot_conn, pos, rotation = 160, horizontal = 20, vertical = -0.13, gripper = 0)
                        pos = move_robot(robot_conn, pos, rotation = 160, horizontal = 0, vertical = 0, gripper = 0)
                        pos = move_robot(robot_conn, pos, rotation = 0, horizontal = 0, vertical = 0, gripper = 0)

                    if class_detected == 1:
                        pos = move_robot(robot_conn, pos, rotation = -3.5, horizontal = 20, vertical = -0.03, gripper = 0)
                        pos = move_robot(robot_conn, pos, rotation = -3.5, horizontal = 20, vertical = -0.03, gripper = 1)
                        pos = move_robot(robot_conn, pos, rotation = 0, horizontal = 0, vertical = 0, gripper = 1)
                        pos = move_robot(robot_conn, pos, rotation = 210, horizontal = 0, vertical = 0, gripper = 1)
                        pos = move_robot(robot_conn, pos, rotation = 210, horizontal = 20, vertical = -0.13, gripper = 1)
                        pos = move_robot(robot_conn, pos, rotation = 210, horizontal = 20, vertical = -0.13, gripper = 0)
                        pos = move_robot(robot_conn, pos, rotation = 210, horizontal = 0, vertical = 0, gripper = 0)
                        pos = move_robot(robot_conn, pos, rotation = 0, horizontal = 0, vertical = 0, gripper = 0)

                    if class_detected == 2:
                        print("Object not recognised. Remove it from the conveyor belt.")
```

Figure 3.20: The robot_control() thread function

3.2.5 Configuration file

```yaml
### YML CONFIGURATION FILE ###    enri

# PC Side connection #
host_pc: "172.20.10.3"
comm_port: 8000
video_port: 5000

# PLC_Side_connection #
host_plc: "192.168.0.241"
plc_port_1: 2000
plc_port_2: 2001

# Camera IDs #
footage_cam_ID: 1
AI_cam_ID: 0

# Path to Keras Model #
AI_model_path: "model/MLModel.keras"
```

Figure 3.21: The configuration file

The configuration file is a YML file that contains the parameters related to:

- IP address and ports of the interface through which the server connects to the client;
- IP address and ports of the interface through which the server connects to the PLC;
- The IDs with which openCV identifies the webcams used by the system (usually ID 0 is reserved for the computer built-in one, so it is necessary to pay particular attention when setting these parameters);
- The path of the Keras model that makes the predictions.

The parameters are automatically taken via the Python yml library, making the system modular (for example, the AI model can be easily replaced with another to perform a different task, simply by creating a new model and passing its path) and easy to configure.

3.2.6 Ladder diagram

To create the system and ensure its correct functioning, modifications were made to the original ladder diagram [4]. In particular:

- a new line was inserted to control the conveyor belt (Figure 3.22), dependent on the output of the proximity sensor. If the proximity sensor detects an object, the conveyor belt is stopped. Once the object has been picked up, the conveyor starts moving again after 3 seconds thanks to the presence of a TOFF timer. Since the output of the sensor is, normally (when there is no object in its detection range), 24 V, a NC relay input was used. When the object is removed from the sensor detection field, the conveyor continues to remain stationary because the TOFF timer keeps the output line high for another 3 seconds;

- a socket (client) block was inserted (Figure 3.23) to allow the PLC to notify the presence of an object to the server machine. The socket connects to the server when the physical start button is pressed;

- a TSEND block (Figure 3.24) was inserted, which sends a "start" string to the server as soon as the StopConveyor output becomes high, i.e. when an object is detected by the proximity sensor. This implements the server notification mechanism;

- a TDISCON block (Figure 3.25) has been inserted to manage the disconnection of the PLC from the server, which is carried out when the physical Stop button is pressed.

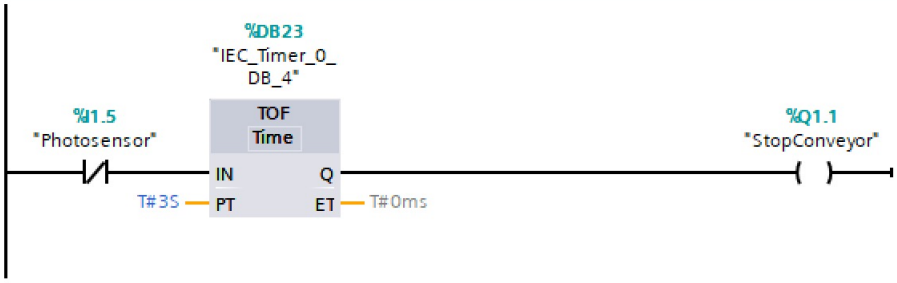


Figure 3.22: Conveyor belt control line

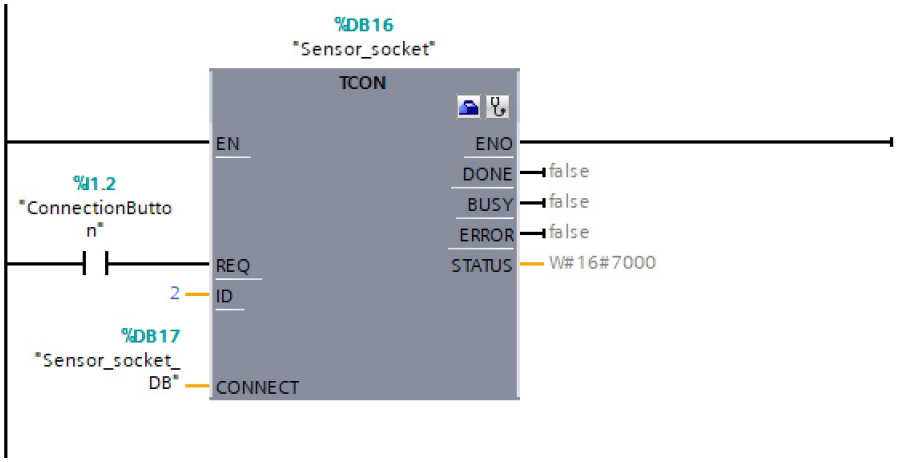


Figure 3.23: Socket block

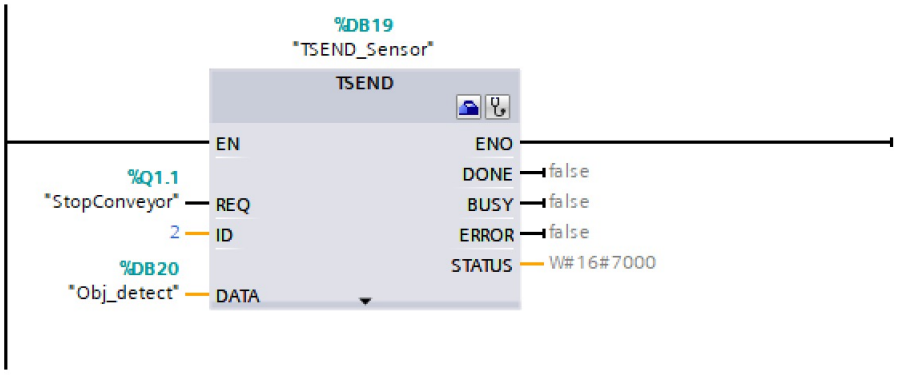


Figure 3.24: TSEND block

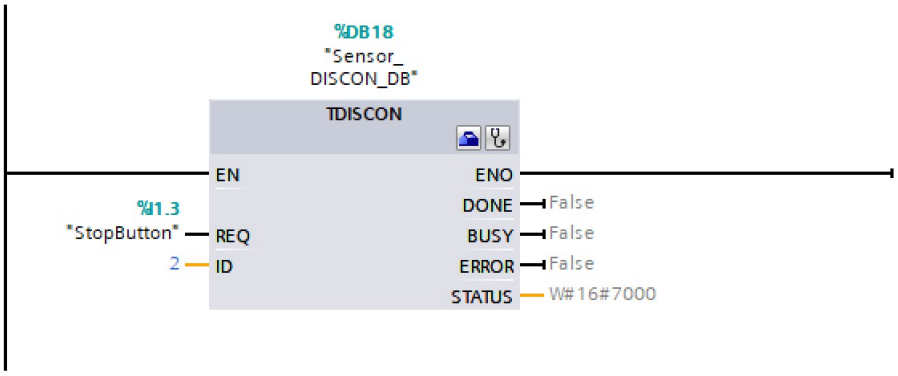


Figure 3.25: TDISCON block

Bibliography

[1] Project Github Repository https://github.com/enrikata/IAAR-Project/tree/main

[2] Previous robot project https://github.com/giorgionocera/fischertechnik-3D-robot/tree/main

[3] Final schematic of the previous robot project https://github.com/giorgionocera/fischertechnik-3D-robot/blob/main/documentation/capitolo3/figure/final_schematic.png

[4] Ladder diagram of the previous robot project https://github.com/giorgionocera/fischertechnik-3D-robot/tree/main/tia%20portal/Project_v3_nuovo

[5] Photoelectric sensor datasheet (E3F2-DS10C4) https://datasheetspdf.com/pdf-file/555259/OmronElectronics/E3F2-DS10C4/1

[6] Dataset generator Github repository https://github.com/elkorchi/2DGeometricShapesGenerator

[7] Matlab code converted in python code https://github.com/giorgionocera/fischertechnik-3D-robot/blob/main/matlab/move_robot.m