UNIVERSITA’ DEGLI STUDI DI MESSINA

Dipartimento di Ingegneria

Ingegneria Elettronica per l’Industria

Smart Green House

Vincenzo Bucaria
Emilia Currò
Aldo Fonseca
Marco Gitto

Prof. Luca Patanè

ANNO ACCADEMICO 2023/2024

Index

1. Introduction

In modern times the work of industrialization proceeds unabated, extending into a wide variety of fields and applications. Agriculture represents one of the areas where this innovation produces results of wide interest and can demonstrate how automation can be of crucial importance the organization and management of industrial processes.

The project under analysis in this paper poses as a simulation of an agricultural production environment in a green house, where the management of environmental parameters is carried out in real time in order to always be able to guarantee an optimal level of plant development condition.

In addition, the system proposes a second level of automation, responsible of the analysis of individual subjects in the measurement field, i.e. each plant in the green house, to provide for its sustenance (through an irrigation system), and to proceed to the harvesting stage as soon as the conditions are reached.

Thus, the smart greenhouse is a smart system in which the growth of plants and the harvesting of their fruits is managed in total autonomy.

The case study examined focuses on tomato plants; thus, the system will specifically recognize "tomatoes" and assess their degree of ripeness.

When the algorithm recognizes a sufficient degree of ripeness, the system will proceed to harvest.

The greenhouse adopts a series of sensors to ensure that the system operates automatically, focusing on the growth and harvesting of vegetables. The simulated environment under examination aims to be able to produce and develop over a long duration, hopefully over the entire year, by monitoring the conditions and managing them in order to keep the seedlings healthy, and lead to the ultimate goal of harvesting.

Inside the greenhouse, a conveyor belt oversees setting the seedlings in motion, which are brought to a central control station. The latter is responsible for checking their condition and irrigating them. In case a seedling with a visible tomato is detected, the station proceeds to the stage of harvesting it. Once the operation (whether checking, irrigation, harvesting, or all events) is completed, the conveyor belt is reactivated to proceed to the next seedling. In fact, the conveyor belt forms a loop so that seedlings are checked daily until the final condition is reached. This greenhouse design represents not only an automatic system, but also one of intensive farming, (referring to a small group of plants), which can also result in space saving.

The greenhouse is constituted as follows:

- The greenhouse environment is equipped with sensors and actuators to react to external conditions such as temperature and humidity. In fact, it consists of humidity and temperature sensors, and actuators to adjust these parameters by opening and closing the greenhouse panels and using some fans to cope with excessive humidity. In contrast, when a heating operation is needed, an IR light lamp is used.

- The conveyor belt, equipped with its own motor, takes care of the movement of the seedlings. It also consists of a photocell and a Hall-effect sensor, so that it can signal to the system when a seedling is in front and when a rotation has been completed, respectively.
- The control station consists of a Fischertechnik 4DOF robotic arm, which features a cutter on top of the gripper. It is also equipped with a camera, which deals with the control of the seedling at the level of target detection, namely the tomato. This operation is also carried out thanks to machine learning focused on the datasets referred to tomatoes, provided by "Kaggle."

As anticipated, a photocell takes care of detecting when the seedling has arrived in front of the control station, triggering the signal that stops the conveyor belt, to allow the operations of the station itself.

Also present here is the irrigation system for the seedlings, which provides a small amount of water each time the seedlings pass by.

Finally, the control panel features start and emergency buttons, respectively "Start" of the system, and Emergency, which stops the entire system. The whole thing is, of course, connected to Siemens PLC.

2. Tecnologies and Materials involved

2.1.1 Hardware

As mentioned earlier, the smart greenhouse is composed of various systems, sensors and actuators.

SIEMENS S7-1200 PLC

The center that manages all activities is represented by the PLC.

The PLC (Programmable Logic Controller) represents the best evolution of automation systems at the industrial level, as it allows easy implementation and easy upgrading, unlike the older relay logic systems, resulting also as a "low cost" solution compared to other alternatives.

The flexibility mentioned of being easily upgradable lies in the characteristic of this controller being a real-time type of system that implements logic, sequential, and timer functions.

The S7-1200 PLC manufactured by Siemens was used in this simulation; exactly the 1215C AC/DC/RLY CPU version was employed in this project.

It provides 14 24V digital inputs, 10 relay outputs, 2 analog inputs and 2 analog outputs.

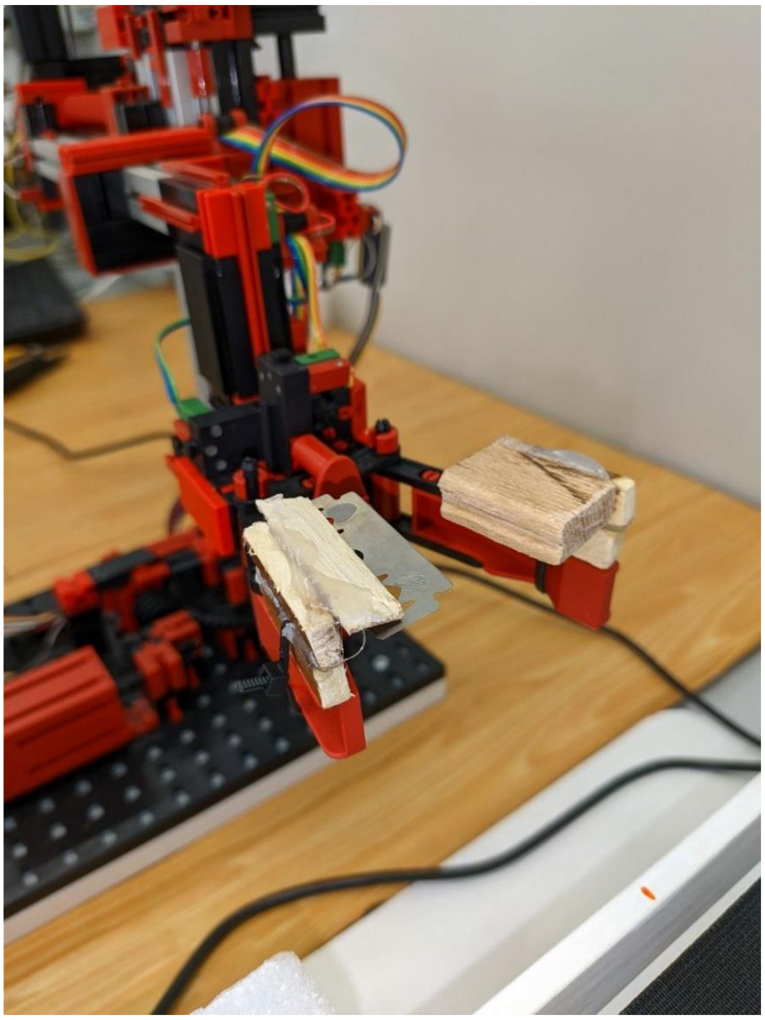
The programming of the PLC takes the wired-logic description method and readjusts it to today’s components using ladder diagram language.

As usual, the SFC diagram was also drawn up at the design stage, which facilitates the task of drawing the ladder diagram itself, as it can in fact more easily indicate the various components of the state machine to be defined.

FISCHERTECHNIK 3D-ROBOT 24V ROBOTIC ARM

The "Fischertechnik 3D-Robot 24V" robotic arm, is a 3DOF arm (3 degrees of freedom), which specifically consists of a revolution joint (to be able to rotate around the fulcrum) and two prismatic joints, which allow movements on the vertical axis, and on the orthogonal of the latter.

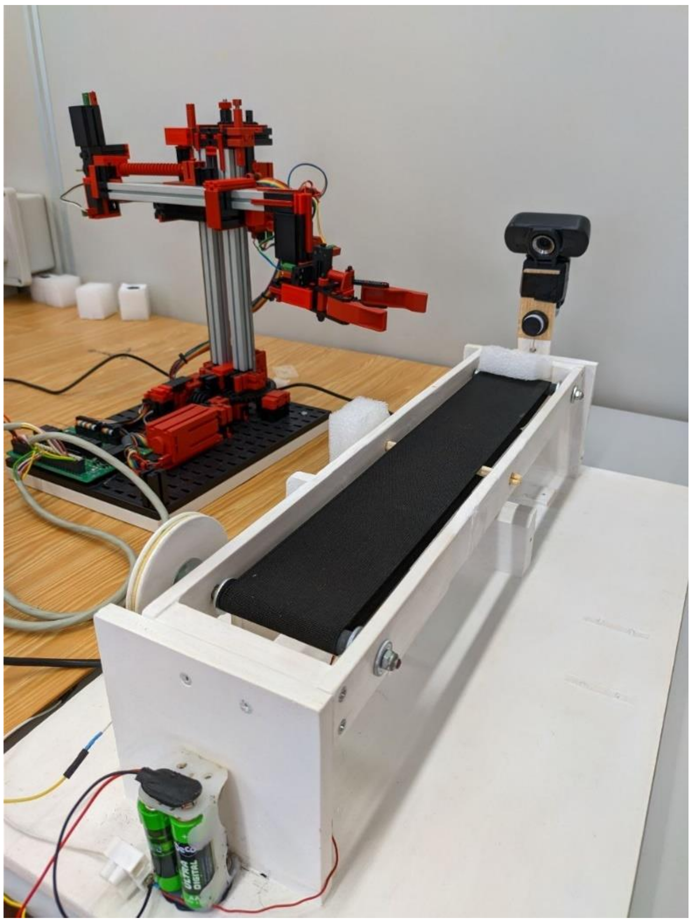
The gripper represents the end effector of the robotic arm.



Also implemented on the gripper is the cutter, which in this simulation represents the section of the robotic arm that performs the collection operation. The cutter was attached to the plastic structure of the gripper through salvaged materials such as industrial cable ties, and wooden boards in view, as always, of resource conservation.

CONVEYOR BELT

The simulation targets, tomato seedlings, are to be moved through the chain of control mentioned earlier. This task is performed by a conveyor belt. In the industrial field, assembly lines are the main example of conveyor belt application, which appear in many different configurations to address a wide variety of transportation needs.



In this project, the conveyor belt takes place on a linear route, running along a loop track, since as seen above; the targets need to be taken care multiple times during their life span, toward fruit ripening. Since it is therefore necessary for them to receive irrigation and monitoring multiple times, the ring shape was the option that was opted for.

In simulation, a conveyor belt of minimal size was used, which is well configured with the Fischertechnik robotic arm in terms of size. On this belt about 6 cm wide and 50 cm long, spheres comparable in size to a tomato and tomatoes themselves can be placed. A test was, however, carried out through agricultural jars of that could fit within the width dimensions of the tape, to simulate a chain of plants, even if the jar was empty (given the impossibility of reproducing a realistic situation congruent with the size of the system available in the laboratory).

The conveyor used is equipped with a DC brush motor, and connected to the belt through a rubber band and two wheels that perform the traction.

PHOTOELECTRIC SENSOR AND WEBCAM

Working closely with the conveyor belt are the photoelectric sensor and the webcam.

The first interfaces with the PLC as an input, which causes the conveyor belt to stop at the right time for the robotic arm to perform its operations.

The sensor was placed sideways to the conveyor belt, so that each individual seedling is detected distinctly from the others and occupying the passage area for the shortest possible time.

Instead, the camera is connected to a pc; this performs the operation of recognizing the tomato itself and checking its state of ripeness. The camera was placed on the same axis as the photoelectric sensor to perform the same range of visibility. A Logitech webcam, already located in lab, was used.

2.1.2 Simulation Level

The section concerning the management of the greenhouse and its environmental parameters was simulated. This decision was made because the laboratory's PLC S7 1215C does not have sufficient inputs and outputs to concurrently manage all components related to both the greenhouse and the Fischertechnik robotic arm.

GREEN HOUSE SENSORS

- DHT11 Temperature and Humidity Sensor: Integrated sensor.
- Arduino UNO: Used to condition signals from the temperature sensor to simplify ladder programming and utilize digital inputs on the PLC.
- Hall Effect Sensor: Used to detect the completion of the control cycle. A magnet placed at the beginning of the conveyor belt triggers the Hall effect sensor upon returning to that position, signaling the completion of the control cycle.
- Relays: Used at Arduino outputs to adhere to the 24V standards used by the PLC.

GREEN HOUSE SENSORS ACTUATORS

- Panel Opening Pistons: Also known as "rod actuators," costing approximately €80 each. These actuators move greenhouse panels to facilitate internal ventilation, thereby lowering the temperature.
- IR Lamps: Used when the temperature is below normal to heat the environment. Typically cost around €10-15 each.
- Fans: These fans circulate air to lower both temperature and humidity levels. Unit costs range from €200 to €300.

2.1.3 Software

For the design of the smart greenhouse, two primary software tools were utilized:

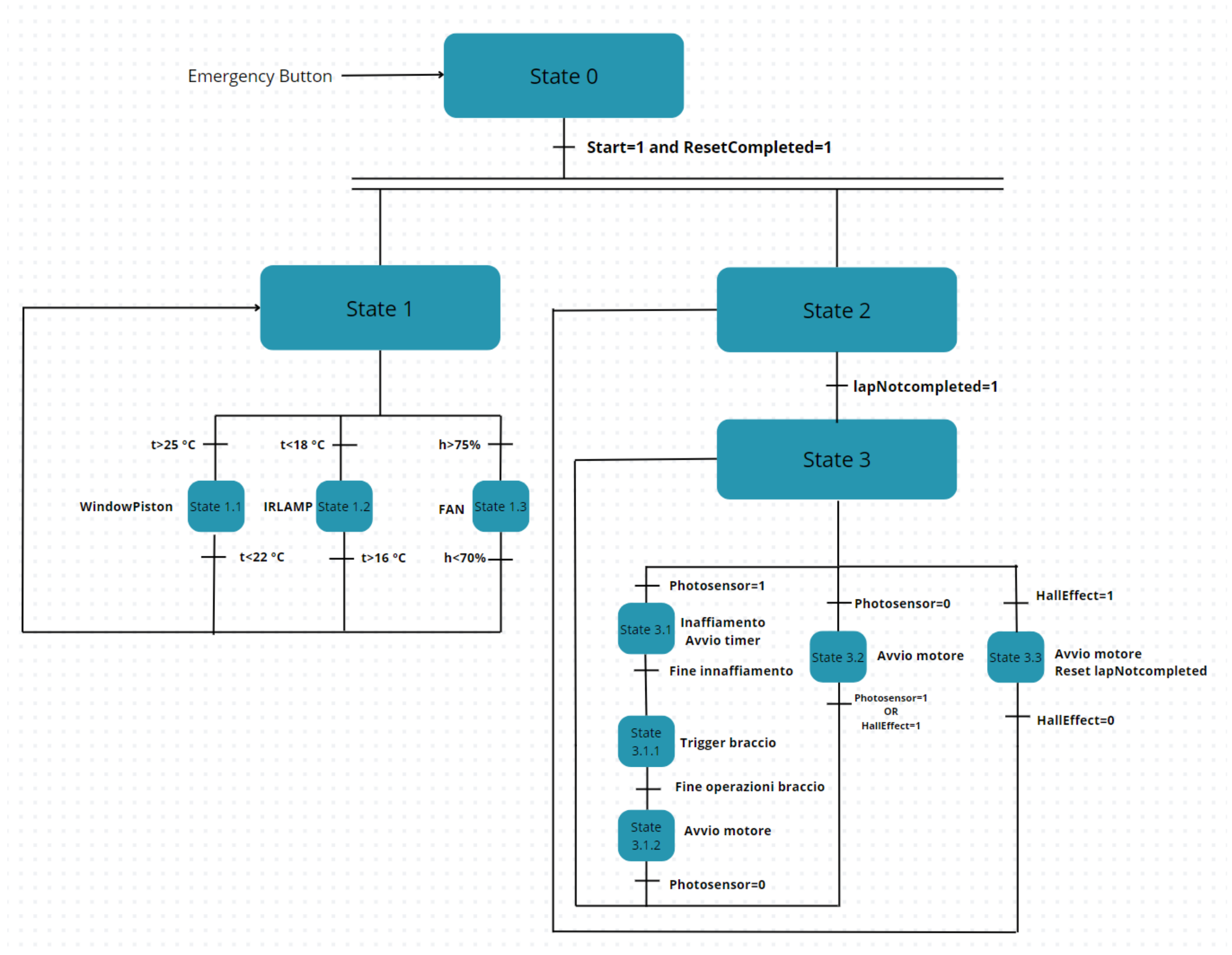
- Siemens TIA Portal V16: Used for writing PLC logic, programming, and interfacing with all inputs and outputs connected to the PLC. TIA Portal is specifically designed for PLC programming in ladder logic. At the project outset, defining the desired PLC, CPU, and version is crucial. Establishing input and output tables facilitates linking with project components, enabling smooth program execution. Simulation of ladder diagrams aids in identifying any deficiencies through input state analysis, ensuring connections are correctly operational on the PLC or forced by the user for system behavior testing.

- Python: This open-source high-level language was employed for writing scripts that identify tomatoes at the correct ripeness using machine learning based on open-source datasets relevant to the project. The Python script also interfaces with the robotic arm to define movements across its various axes.

3. Implementation

3.1 PLC implementation

The implementation of the PLC-based section of the project begins through the drafting of the corresponding Sequential Flow Chart (SFC). The drafting of the SFC then allowed the ladder diagram to be drawn in a simple, clear and readable way.



As can be seen from the figure, the diagram is composed, in addition to the initialization state, of two macro-sections executed in parallel.

- The section on the left (section A) is related to the management of environmental parameters, that is, ambient temperature and humidity, to have within the greenhouse a climate suitable for the proper growth of the fruits.
- The section on the right (section B) is related to the proper management of the conveyor belt whose task is to bring the seedlings in front of the robotic arm at the most appropriate time.

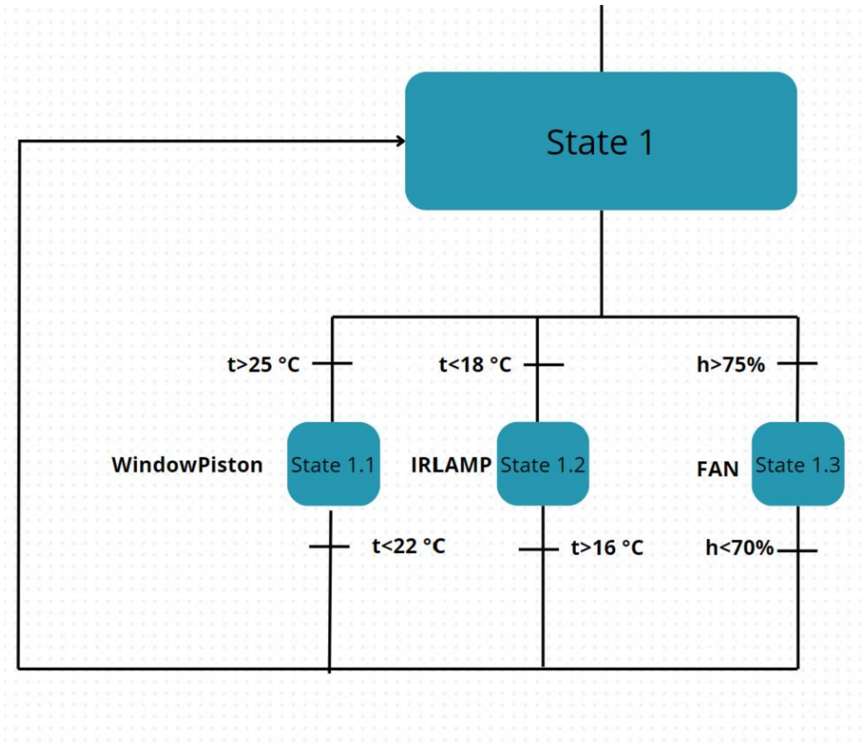
The initialization state, or State 0, describes the behavior of the system in the emergency or initialization phase. As an emergency state, it must be accessible whatever the current state is running. Once in state 0, normal execution can begin or resume upon pressing the start key.

3.1.1 SFC section A

The objective of managing the internal environment of the greenhouse is, as easily understandable, to maintain the most favorable conditions possible for the growth of the plants inside it. To do this, the previously listed sensors and actuators have been used to maintain the following targets:

- Temperature between 18 and 25 degrees
- Humidity between 70% and 75%

When one of these values is no longer within the predetermined range, the system autonomously brings the environment back to a balanced condition by activating one or more actuators.



Specifically, it is possible to observe that section A is composed of three logical states:

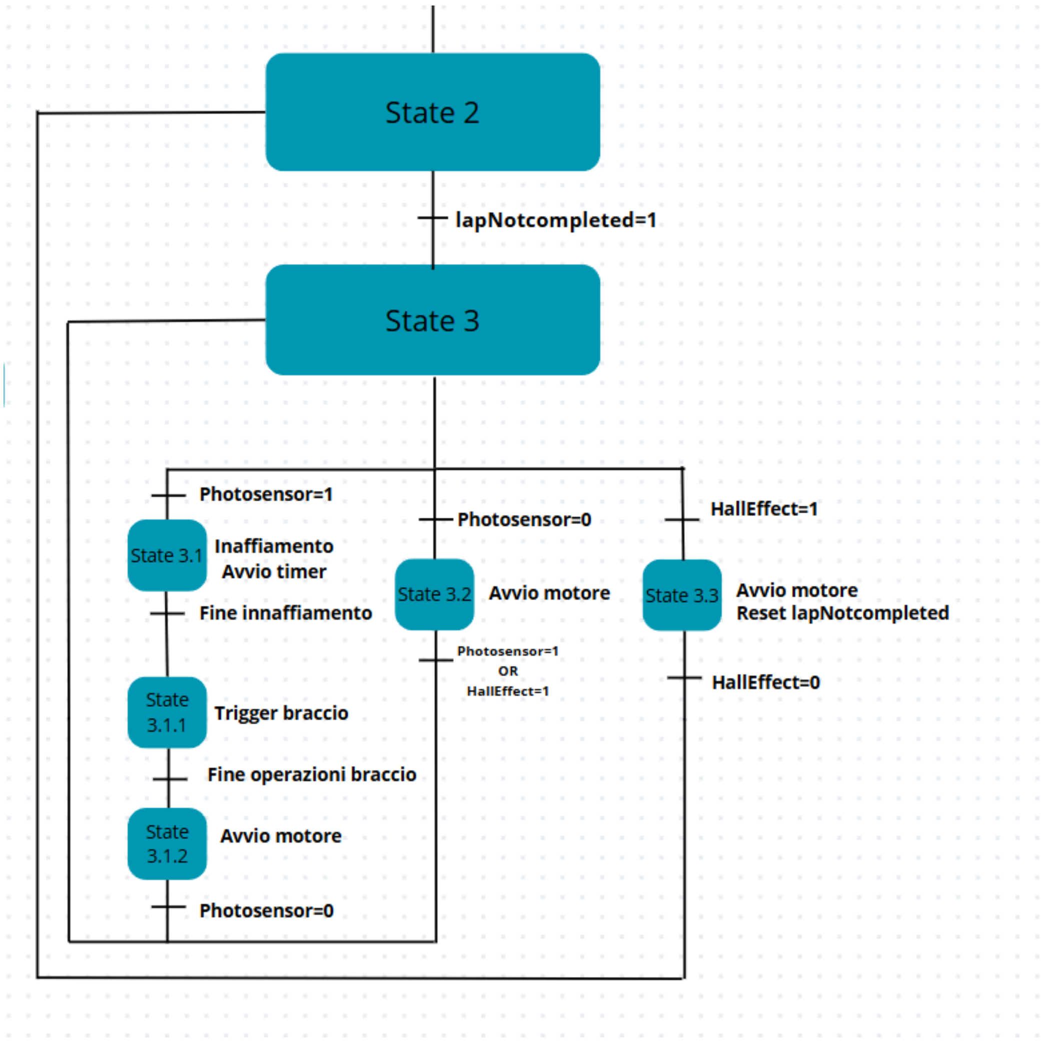
- State 1.1: The system activates this state when it is out of the emergency state (state 0) and the temperature recorded by the temperature sensors is too high (T > 25°). Specifically, in the case of high temperatures (>25), an electro-actuated piston will open a window simultaneously with the activation of a fan to promote air circulation and reduce temperatures. The state exits when T < 22°C.

- State 1.2: The system activates this state when it is out of the emergency state and the temperature recorded by the sensors is too low (T < 18°). The temperature is raised using infrared lamps. The state exits when T > 22°C.
- State 1.3: The system activates this state when the recorded humidity is too high, that is, when it exceeds 75%. The state exits when the humidity percentage is below 70%.

It should be noted that states 1.1 and 1.2 are mutually exclusive and that a hysteresis cycle is present to limit uncontrolled access.

3.1.2 SFC section B

As previously mentioned, section B of the sequential flow chart is related to the management of the conveyor belt and communication through the Python server where the recognition software for the correct handling of the fruits is executed.



As shown in the figure, at the top of the logic in section A is **state 2**, which is the idle state. The system remains in this state when it is not necessary to move the conveyor belt (noting that the plants are subjected to analysis and irrigation only once a day, usually at 6:00 PM).

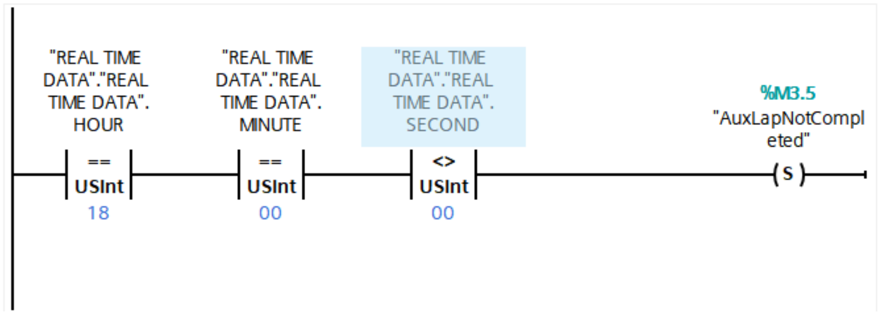
The system exits the idle state and enters state 3 when it is necessary to move the plants, i.e., when it is past 6:00 PM and the daily inspection cycle has not yet been completed. This is controlled by the variable `lapnotcompleted`, which ensures that the plants are moved once a day even if the movement was interrupted or never started due to any emergencies. The management of this variable will be discussed later.

In state 3, the initial decision-making controls for starting or stopping the conveyor belt and initiating the analysis by the server, as well as returning to the idle state, are carried out. More precisely:

- From state 3, the system transitions to state 3.1 if a plant is in front of the analysis station, i.e., when the photosensor detects the plant.
- Once in state 3.1, the system enters state 3.1.1 and stops the conveyor belt, starts irrigation for a few seconds, and then enters state 3.1.2 and notifies the server that a plant is ready for inspection. Once the inspection is completed and any fruit harvesting operations are done, the server notifies the system, and it transitions out of state 3.1.2. The system then starts the motor to move the plant out of the photosensor detection area, after which it exits state 3.1.2 and returns to state 3.
- From state 3, the system transitions to state 3.2 if no plant has yet arrived in the photosensor detection area. During state 3.2, the conveyor belt is running. The system returns to state 3 if the inspection cycle is completed, a condition detected by the Hall effect sensor, or if a plant arrives in the analysis and processing area.
- From state 3, the system returns to state 2 via state 3.3, which is enabled when the input from the Hall effect sensor is true. In state 3.3, the motor is started again to move the magnet and ensure that the Hall effect sensor input is false. Only then does the system exit state 3.3 and return to the idle state.

3.1.3 RTC and lapNotcompleted variable management

Since the system needs to keep track of the time to start the inspection and processing cycle of the plants, the Real Time Clock (RTC) function of the PLC has been used.


"REAL TIME DATA"."REAL TIME DATA". HOUR
==
USInt
18
"REAL TIME DATA"."REAL TIME DATA". MINUTE
==
USInt
00
"REAL TIME DATA"."REAL TIME DATA". SECOND
<>
USInt
00
%M3.5
"AuxLapNotCompleted"
S

Regardless of the system's running state (including in an emergency state), the PLC keeps track of the time, and every day, at exactly 6:00:00 PM, the AuxLapNotCompleted marker is set, which corresponds to the `lapNotcompleted` variable described in the SFC, thus taking the value True.

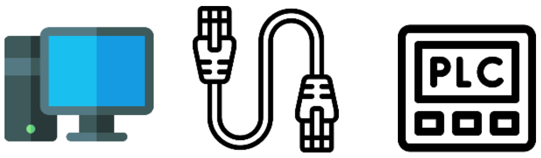
When the `lapNotcompleted` variable is True, if the system is not in an emergency state (State 0), there is a mandatory transition from state 2 to state 3. The variable can only be reset if the system transitions from state 3.3, meaning if the conveyor belt has made a complete rotation, i.e., all the plants have been inspected.

This ensures the start and completion of the daily inspection and fruit harvesting cycle as it covers the following critical cases:

- The system enters an emergency state before 6:00 PM and exits the emergency state at any time after 6:00 PM.
- The inspection cycle has started but is interrupted due to entering the emergency state.

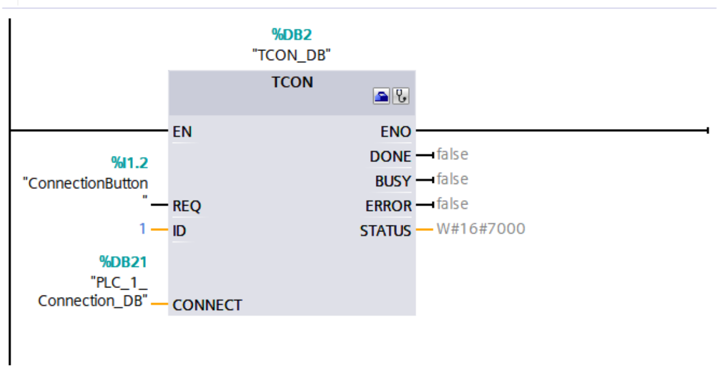
3.1.4 PLC-PC connection

Communication between the PLC and the server is carried out using the TCP/IP protocol, which ensures the reliable transport of information over the network. In fact, the computer and the PLC are directly connected via a local network, thus establishing a peer-to-peer (P2P) connection.

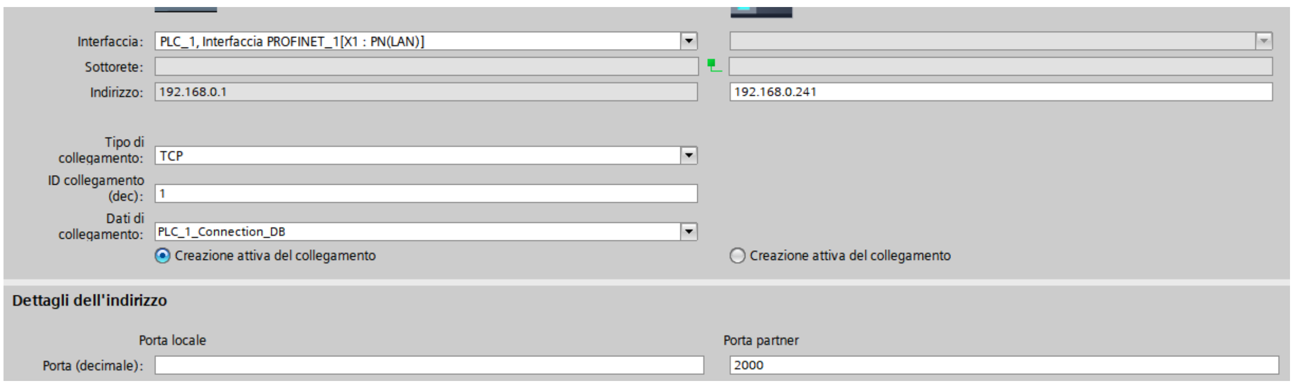


To properly manage communication between the PLC and the server, it is first necessary to configure the PLC's network parameters in TIA Portal.

Subsequently, to allow the PLC to open network sockets and perform the TCP handshake, TCON blocks have been used.



These blocks must be properly configured, specifying the network address and the listening port of the partner (in our case, the Python server). It is also crucial to set the block identifier to assign the socket to other essential network blocks.



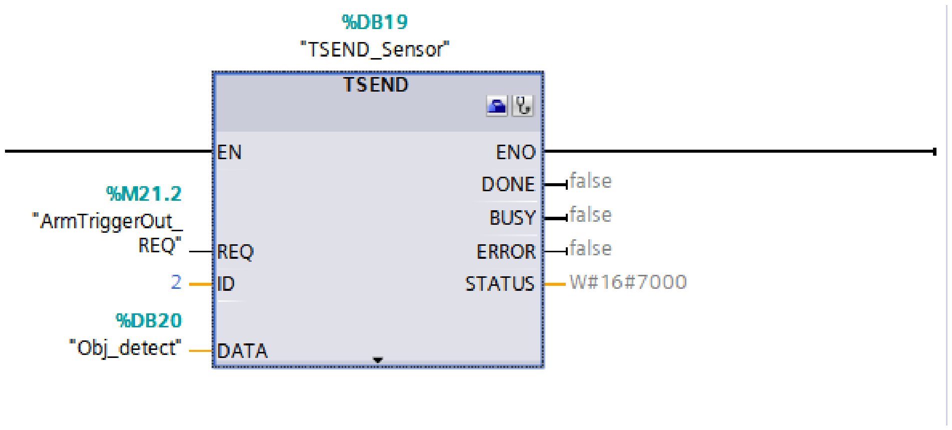
Interfaccia:	PLC_1, Interfaccia PROFINET_1[X1 : PN(LAN)]	
Sottorete:		
Indirizzo:	192.168.0.1	192.168.0.241
Tipo di collegamento:	TCP	
ID collegamento (dec):	1	
Dati di collegamento:	PLC_1_Connection_DB	
	[✓ Creazione attiva del collegamento]	Creazione attiva del collegamento

Dettagli dell'indirizzo

	Porta locale	Porta partner
Porta (decimale):		2000

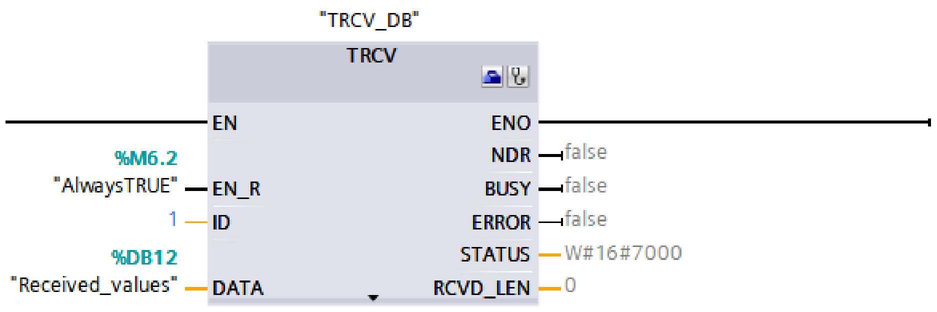
In our system, it was necessary to use two sockets: one for sending and receiving commands over the network for managing the conveyor belt, and another for sending and receiving commands over the network for managing the robotic arm. Both sockets are initialized when the ConnectionButton, equivalent to the system's Start button, is pressed.

For sending data via socket, the TSEND block was used, which is associated with its respective socket via its ID.



For example, the figure shows the TSEND block responsible for sending the message (contained in the Obj_detect database) signaling to the server that a plant is ready to be analyzed. The message is sent every time the ArmTriggerOut_REQ marker changes from false to true, which occurs during the execution of state 3.1.

For receiving messages via socket, the TRCV block was used, also appropriately associated with a socket via its ID.



For example, the figure shows the TRCV block responsible for receiving commands (stored in the Received_values database) for managing the robotic arm.

3.1.5 PLC-side Robot Manipulator Management

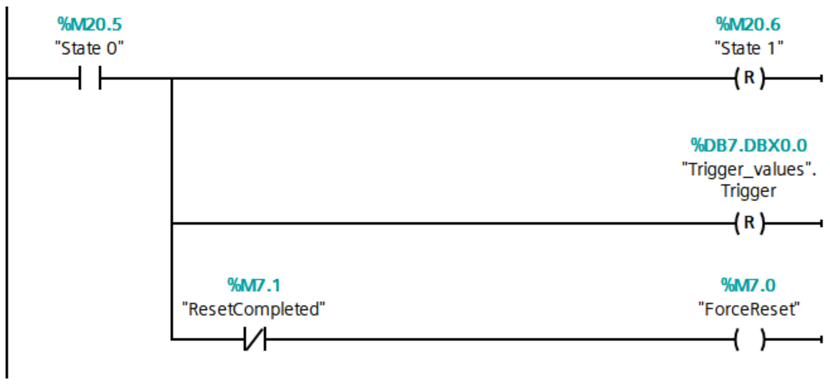
The manipulator used is well-suited for management by a PLC as it operates with a 24V logic. The foundation of the logic is the ladder diagram developed by colleagues in previous years, as it is functional and easily adaptable. We then modified the logic to achieve the desired operation.

More specifically, the ladder implementation allows the robotic arm to be moved upon request from the Python server, provided the system is not in a stop state.

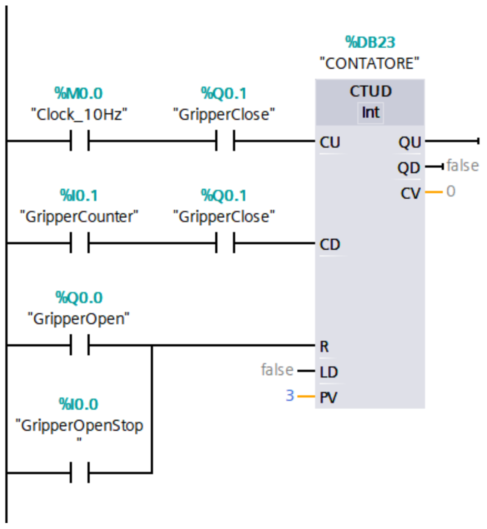
When the server sends the parameters to move the robot over the network, these are collected via one of the two project sockets into the `Received_values` database.

Subsequently, the activation of the coils related to the robotic manipulator actuators depends not only on the various limit switches and counters (connected to encoders) necessary to detect the minimum or maximum excursion of the various manipulator elements but also on the values recorded in the discussed database.

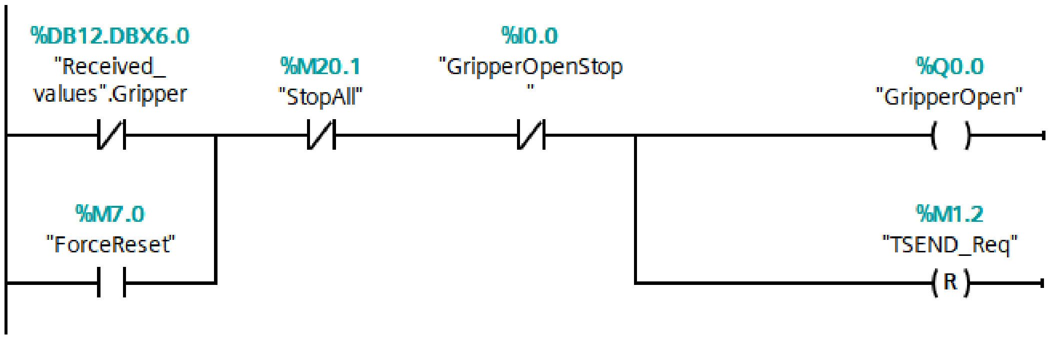
For example, the coil related to the gripper opening is activated when the boolean value in the DB is True; this value is, of course, written to the DB by the Python server. The coil remains activated until the gripper opening reaches its maximum excursion, which is detected by the limit switch connected to the `GripperOpenStop` input.



An essential modification was made to adjust the torque and distance between the gripper appendages during its closing phase, to adapt its grip to the fruit. Considering that the gripper's closure is regulated by a CTUD counter, it was sufficient to change the trigger value by setting it to 3.

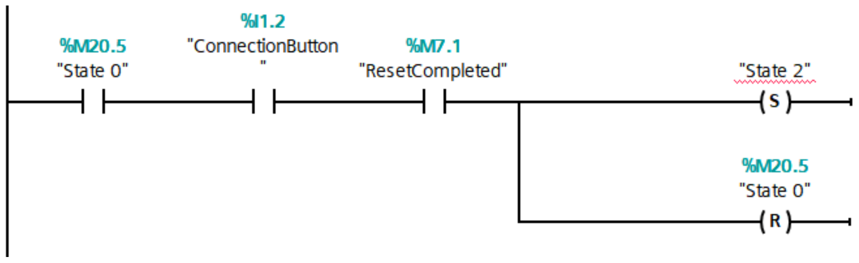


Lastly, efforts were made to allow the start or restart of operations related to the conveyor belt (transition from State 0 to other states) only if the manipulator is in the rest position. For this reason, both when the PLC is powered on and when entering the emergency state (State 0), the



`ForceReset` marker is forced to True (until the reset is completed), with the task of bringing the robotic manipulator to the rest position.

As can be seen, the transition from the rest state to the main state related to section B of the sequential flow chart is allowed only if the `ResetCompleted` marker is True, i.e., when the robotic manipulator is in the rest position.



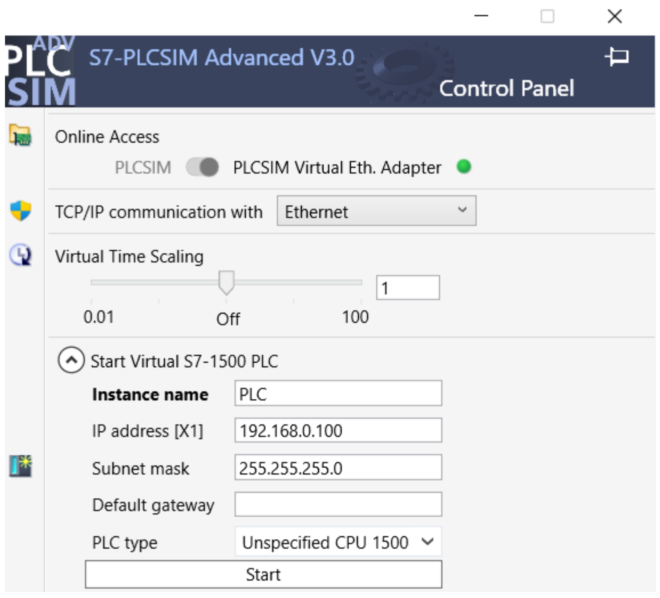
This precaution prevents incorrect handling of the arm; since the Python server has no way of knowing the current position of the end-effector, sending relative movement commands would lead to offset errors.

3.2 Simulation using PLCSIMAdvanced

In the initial design and implementation phases, it was convenient for debugging and understanding to simulate the logic using PLCSIMAdvanced.

It was decided to use PLCSIMAdvanced because, unlike PLCSim, it allows you to virtualize a network interface and therefore allows you to communicate via TCP/IP with the emulated PLC, allowing us to analyze in a simulated environment the synergy between the environment managed by the PLC and the instructions given by the Python server.

It should be noted that PLCSIMAdvanced can only simulate PLCs belonging to the S7-1500 family, so it was necessary to create an additional project on TIA Portal using a different CPU (precisely, of the 1500 family) from the one available in the laboratory


S7-PLCSIM Advanced V3.0
Control Panel

Online Access
PLCSIM ● PLCSIM Virtual Eth. Adapter ●

TCP/IP communication with: Ethernet

Virtual Time Scaling: 1 (0.01 – Off – 100)

Start Virtual S7-1500 PLC

Instance name	PLC
IP address [X1]	192.168.0.100
Subnet mask	255.255.255.0
Default gateway	
PLC type	Unspecified CPU 1500

3.3 Python Scripts

The section dedicated to the Python scripts developed for the project includes the machine learning part for recognizing ripe tomatoes, the management of the robotic arm for harvesting, and the communication between the Server and Client. Below are the details of the individual components.

3.3.1 Machine learning

The machine learning code used for the recognition of ripe tomatoes is contained in the file `train_and_evaluate_classifier.py`. This script handles the training of a classification model based on a dataset of tomato images. Several open-source Python libraries are used, including scikit-learn for modeling and numpy for data manipulation.

3.3.1.1 Libraries

· **numpy**: for efficient manipulation of data arrays.
· **pandas**: for dataset management.
· **scikit-learn**: for creating and evaluating machine learning models.
· **matplotlib**: for data visualization.
· **joblib**: for saving and loading trained models.
· **optuna**: for optimizing hyperparameters.

3.3.1.2 Software Descritiption

The `train_and_evaluate_classifier.py` code performs the following main operations:

- Loading and pre-processing the image dataset.
- Splitting the dataset into training and test data.
- Searching for optimal hyperparameters using Optuna.
- Training a Random Forest classifier with the optimal hyperparameters.
- Evaluating the model's performance on the test data.

The "Kaggle" dataset is loaded using the pandas library. Images are represented as arrays of pixels, and each image has an associated label indicating whether the tomato is ripe or unripe (green). After loading, the data is split into two sets: one for training (80%) and one for testing (20%). This step is crucial for evaluating the model's ability to generalize to unseen data during training.

Optimal Hyperparameters

To optimize the model's performance, the Optuna library is used for hyperparameter tuning. This library implements an automatic optimization process using advanced search techniques to find the optimal combination of model hyperparameters.

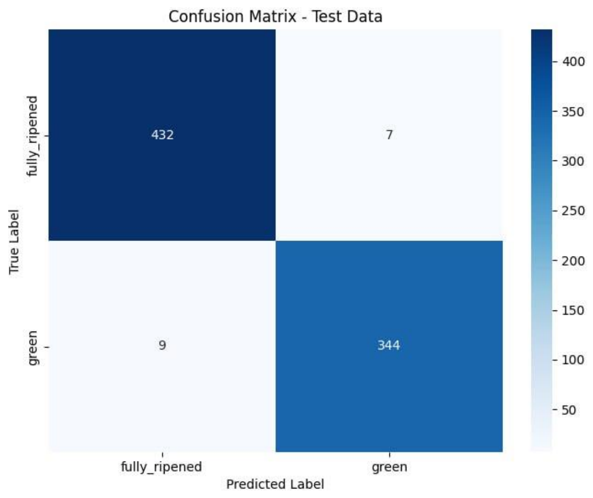
```
import optuna
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score

def objective(trial):
    n_estimators = trial.suggest_int('n_estimators', 100, 500)
    max_depth = trial.suggest_int('max_depth', 10, 50)
    min_samples_split = trial.suggest_int('min_samples_split', 2, 20)
    min_samples_leaf = trial.suggest_int('min_samples_leaf', 1, 20)

    rf_classifier = RandomForestClassifier(
        n_estimators=n_estimators,
        max_depth=max_depth,
        min_samples_split=min_samples_split,
        min_samples_leaf=min_samples_leaf,
        random_state=42
    )
    rf_classifier.fit(X_train, y_train)
    accuracy = rf_classifier.score(X_test, y_test)

    return accuracy

study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100)
best_params = study.best_params
```

MODEL TRAINING

Once the optimal hyperparameters are obtained, the Random Forest model is trained using these parameters. The model training is performed through the `fit()` function of the scikit-learn library, which optimizes the model parameters to minimize the classification error on the training data.

MODEL EVALUATION

To evaluate the model, precision, recall, and F1-score are used as metrics, which provide a measure of the c model's performance. Additionally, a confusion matrix is generated to visualize the number of correct and incorrect classifications.



TRAINING RESULTS

The machine learning model achieved the following results on the test data:

	PRECISION	RECALL	F1-SCORE	SUPPORT
FULLY_RIPENED	0.9795	0.9795	0.9795	439
GREEN	0.9745	0.9745	0.9745	353
ACCURACY	0.9773			792

Model Performance Evaluation

The trained Random Forest model showed a high accuracy of 97.73% on the test data, with almost identical accuracy, recall, and f1-score for both classes (ripe and green tomatoes). This indicates that the model is highly effective at distinguishing between ripe and green tomatoes, with a low error rate.

The metrics used to evaluate model performance are defined as follows:

- **Accuracy**: The proportion of correct predictions out of the total number of predictions made.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Precision**: The proportion of correct predictions among all predictions made for a certain class.

$$Precision = \frac{TP}{TP\ +\ FP}$$

- **Recall**: The proportion of correct predictions across all actual instances of a certain class.

$$Recall\ = \frac{TP}{TP + FN}$$

- **F1-Score**: the harmonic means of precision and recall.

$$F1 - Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

3.3.2 Robot Manipulator Management

Robotic arm management is implemented in thread_webcam.py and move_robot.py scripts. These scripts take care of managing the webcam for tomato detection and controlling the robotic arm for the harvesting operation, respectively.

3.3.2.1 Description of the thread_webcam.py code

The thread_webcam.py script uses the OpenCV library for webcam management and tomato detection. The captured image is processed in real-time to identify the tomatoes and determine if they are ripe for harvesting.

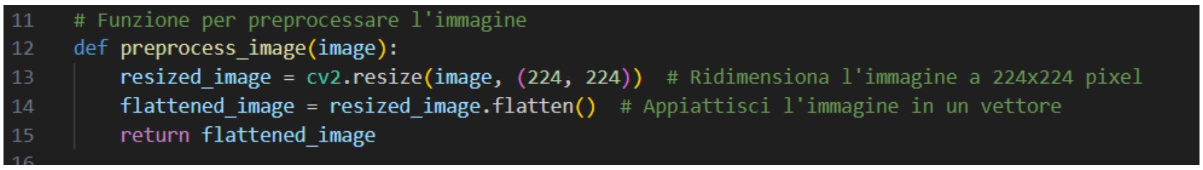
Main Libraries Used

1. **OpenCV:** for image acquisition and processing.
2. **base64**: for image encoding.
3. **numpy**: For manipulating data arrays.
4. **Pandas**: for managing datasets.
5. **Joblib**: For loading the trained model.
6. **Threading**: For handling threading.
7. **Socket**: For communication between processes.

Portion of the code

The following are the most important sections of the code used for webcam management and tomato recognition.

preprocess_image() function

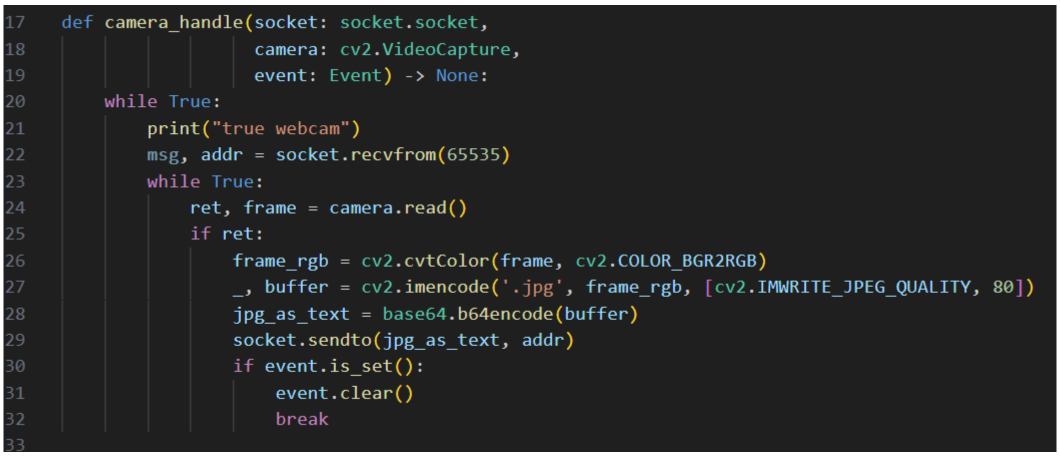


```python
# Funzione per preprocessare l'immagine
def preprocess_image(image):
    resized_image = cv2.resize(image, (224, 224))  # Ridimensiona l'immagine a 224x224 pixel
    flattened_image = resized_image.flatten()  # Appiattisci l'immagine in un vettore
    return flattened_image
```

This function scales the captured image to 224x224 pixels and flattens it into a vector, preparing it for input to the machine learning model.

camera_handle() function

This feature manages the video stream from the webcam and sends the captured frames via sockets. The frame is converted to RGB, base64 encoded, and sent to the recipient.

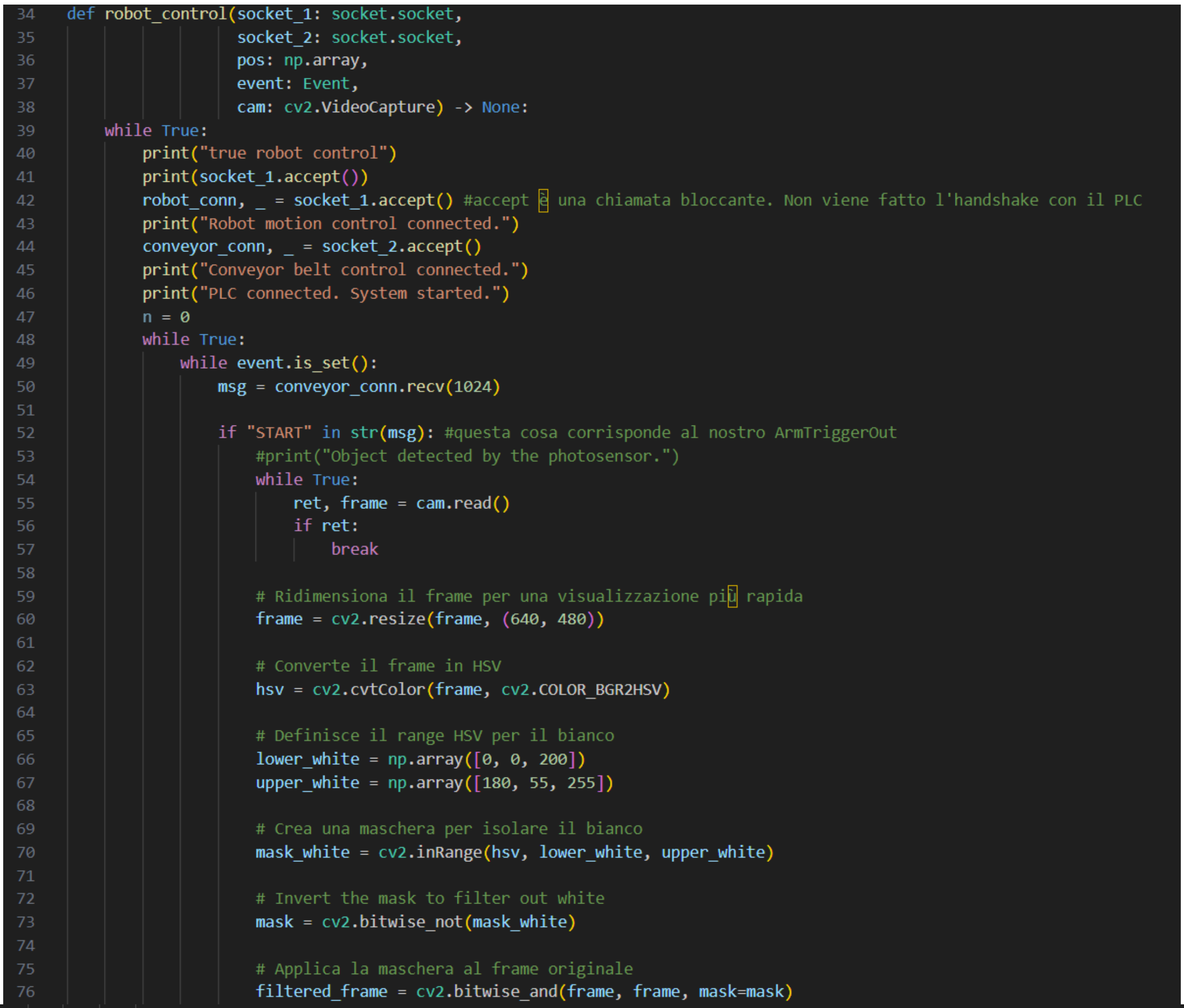


```python
def camera_handle(socket: socket.socket,
                  camera: cv2.VideoCapture,
                  event: Event) -> None:
    while True:
        print("true webcam")
        msg, addr = socket.recvfrom(65535)
        while True:
            ret, frame = camera.read()
            if ret:
                frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                _, buffer = cv2.imencode('.jpg', frame_rgb, [cv2.IMWRITE_JPEG_QUALITY, 80])
                jpg_as_text = base64.b64encode(buffer)
                socket.sendto(jpg_as_text, addr)
                if event.is_set():
                    event.clear()
                    break
```

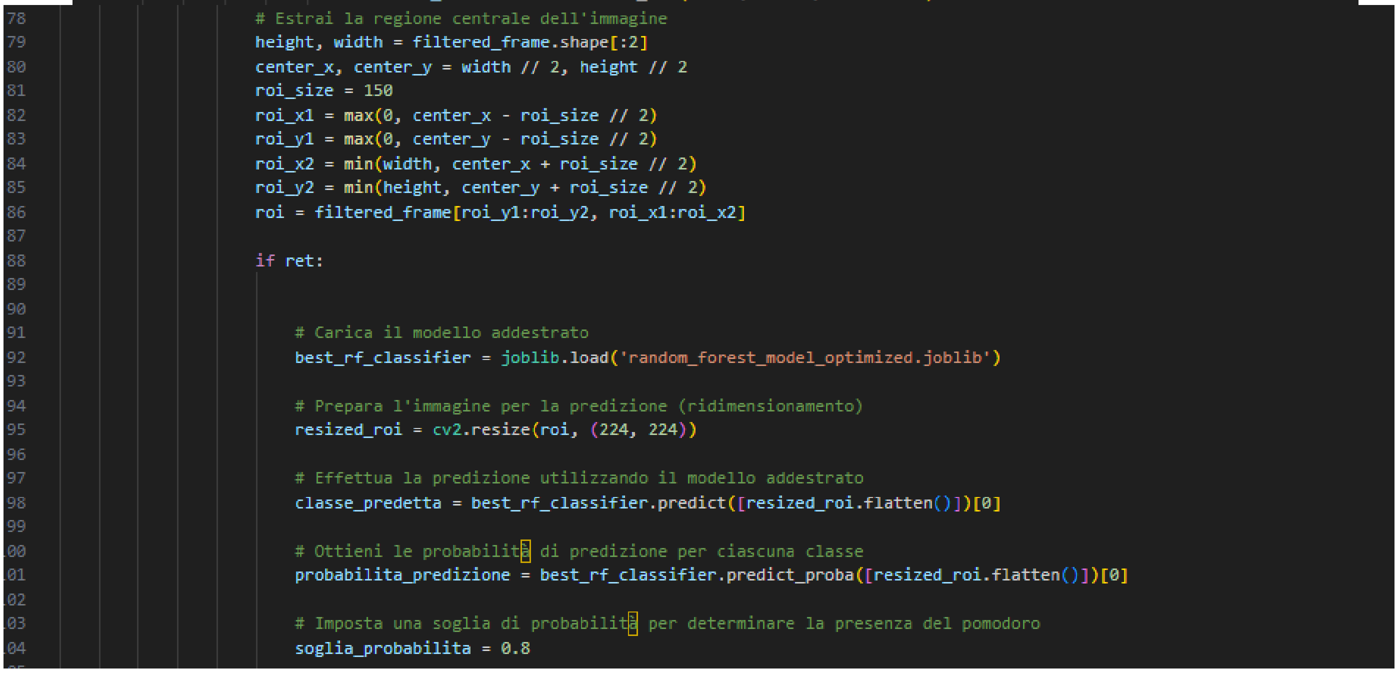
robot_control() function

This function manages the control of the robotic arm and conveyor belt. It uses the webcam to capture images, process the frames by applying a white color elimination filter since the trained model's images showed the tomato on a white background; therefore, showing white objects to the webcam could result in prediction errors. To detect the presence of tomatoes, first observe processed frames to detect the presence of tomatoes, first observing whether a tomato is either ripe or green using the trained model, and then checks whether this detected tomato is ripe.

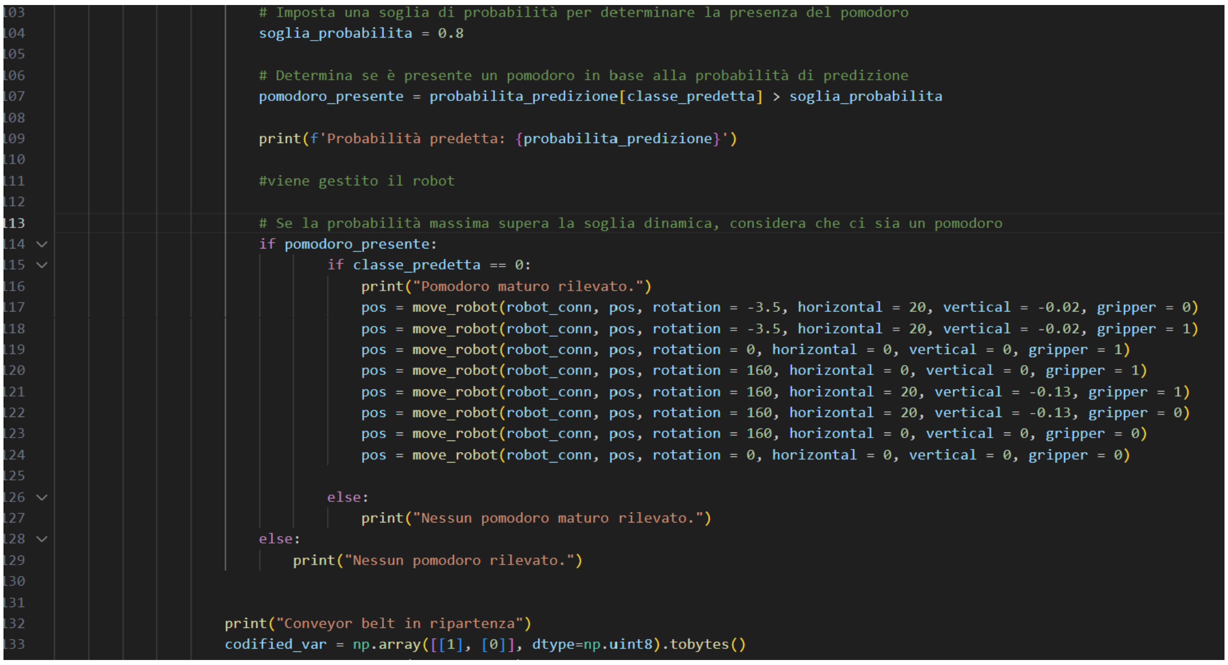
If a ripe tomato is detected, the robotic arm is operated to pick it up.



```python
def robot_control(socket_1: socket.socket,
                  socket_2: socket.socket,
                  pos: np.array,
                  event: Event,
                  cam: cv2.VideoCapture) -> None:
    while True:
        print("true robot control")
        print(socket_1.accept())
        robot_conn, _ = socket_1.accept() #accept è una chiamata bloccante. Non viene fatto l'handshake con il PLC
        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): #questa cosa corrisponde al nostro ArmTriggerOut
                    #print("Object detected by the photosensor.")
                    while True:
                        ret, frame = cam.read()
                        if ret:
                            break

                    # Ridimensiona il frame per una visualizzazione più rapida
                    frame = cv2.resize(frame, (640, 480))

                    # Converte il frame in HSV
                    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

                    # Definisce il range HSV per il bianco
                    lower_white = np.array([0, 0, 200])
                    upper_white = np.array([180, 55, 255])

                    # Crea una maschera per isolare il bianco
                    mask_white = cv2.inRange(hsv, lower_white, upper_white)

                    # Invert the mask to filter out white
                    mask = cv2.bitwise_not(mask_white)

                    # Applica la maschera al frame originale
                    filtered_frame = cv2.bitwise_and(frame, frame, mask=mask)
```



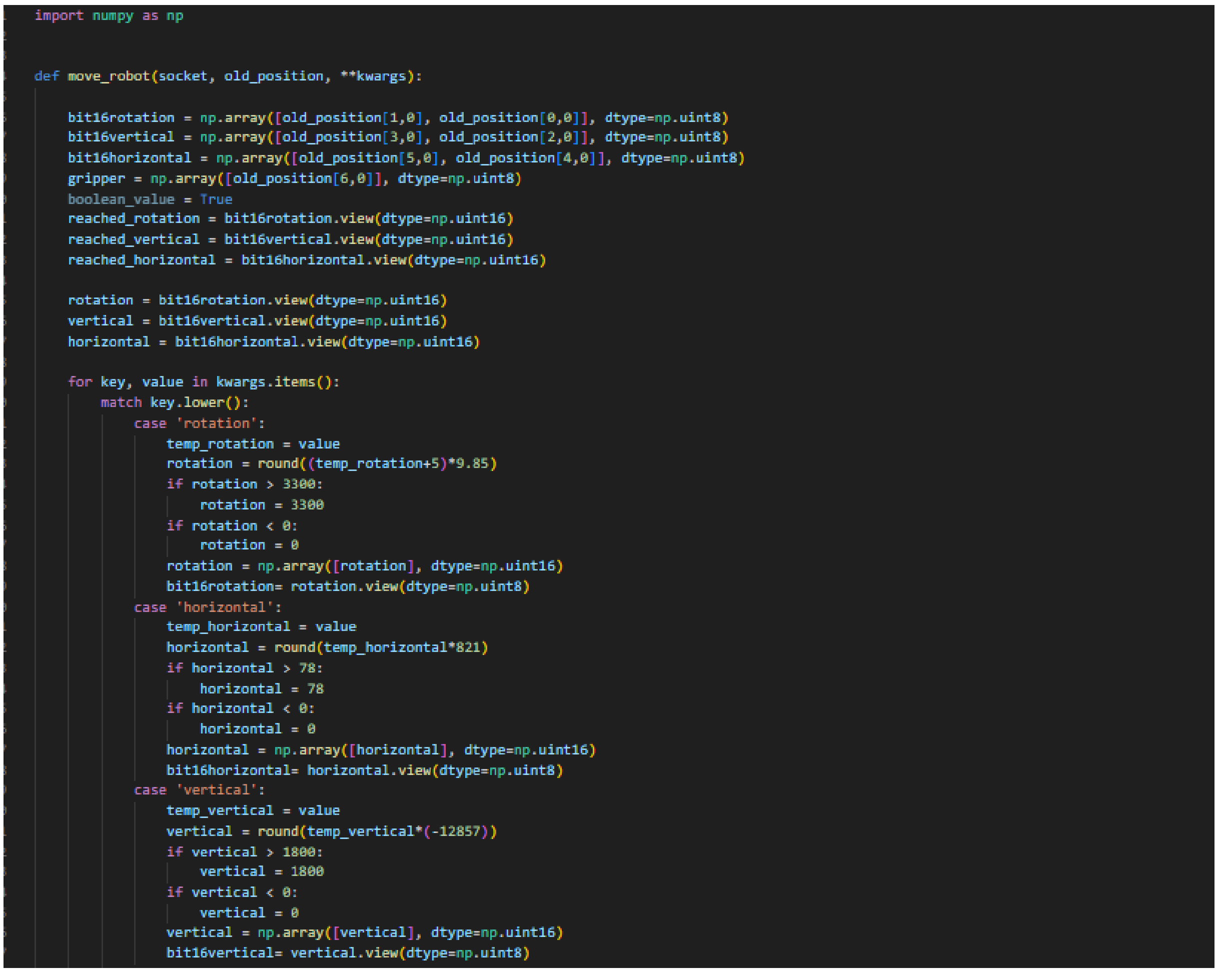
```python
                    # Estrai la regione centrale dell'immagine
                    height, width = filtered_frame.shape[:2]
                    center_x, center_y = width // 2, height // 2
                    roi_size = 150
                    roi_x1 = max(0, center_x - roi_size // 2)
                    roi_y1 = max(0, center_y - roi_size // 2)
                    roi_x2 = min(width, center_x + roi_size // 2)
                    roi_y2 = min(height, center_y + roi_size // 2)
                    roi = filtered_frame[roi_y1:roi_y2, roi_x1:roi_x2]

                    if ret:


                        # Carica il modello addestrato
                        best_rf_classifier = joblib.load('random_forest_model_optimized.joblib')

                        # Prepara l'immagine per la predizione (ridimensionamento)
                        resized_roi = cv2.resize(roi, (224, 224))

                        # Effettua la predizione utilizzando il modello addestrato
                        classe_predetta = best_rf_classifier.predict([resized_roi.flatten()])[0]

                        # Ottieni le probabilità di predizione per ciascuna classe
                        probabilita_predizione = best_rf_classifier.predict_proba([resized_roi.flatten()])[0]

                        # Imposta una soglia di probabilità per determinare la presenza del pomodoro
                        soglia_probabilita = 0.8
```



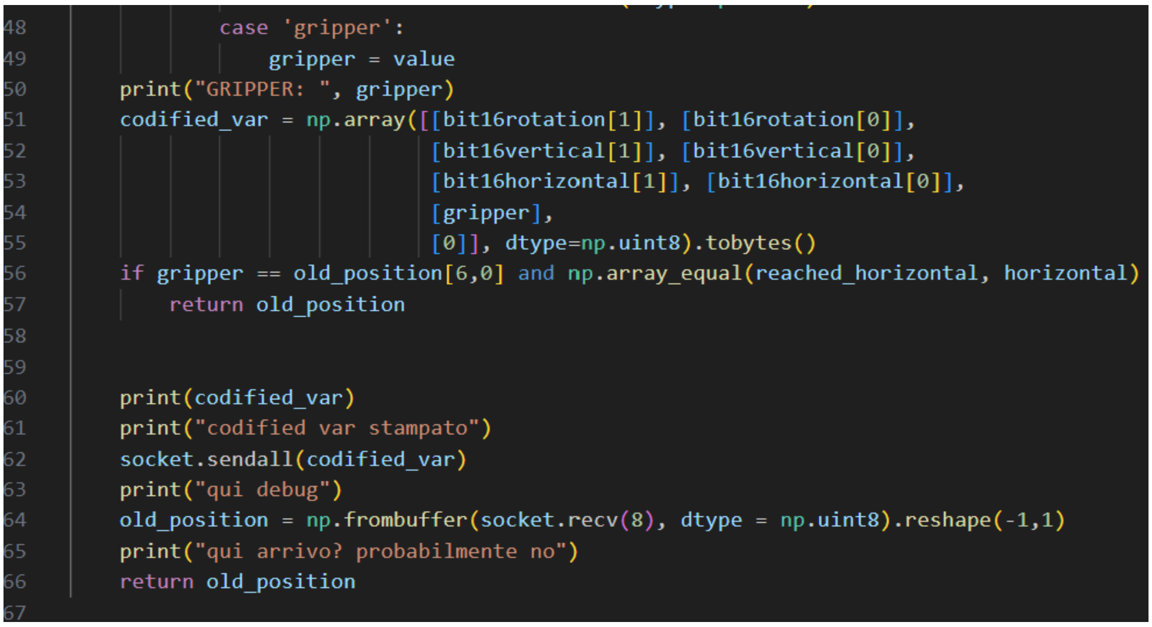
```python
    # Imposta una soglia di probabilità per determinare la presenza del pomodoro
    soglia_probabilita = 0.8

    # Determina se è presente un pomodoro in base alla probabilità di predizione
    pomodoro_presente = probabilita_predizione[classe_predetta] > soglia_probabilita

    print(f'Probabilità predetta: {probabilita_predizione}')

    #viene gestito il robot

    # Se la probabilità massima supera la soglia dinamica, considera che ci sia un pomodoro
    if pomodoro_presente:
        if classe_predetta == 0:
            print("Pomodoro maturo rilevato.")
            pos = move_robot(robot_conn, pos, rotation = -3.5, horizontal = 20, vertical = -0.02, gripper = 0)
            pos = move_robot(robot_conn, pos, rotation = -3.5, horizontal = 20, vertical = -0.02, 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)

        else:
            print("Nessun pomodoro maturo rilevato.")
    else:
        print("Nessun pomodoro rilevato.")


print("Conveyor belt in ripartenza")
codified_var = np.array([[1], [0]], dtype=np.uint8).tobytes()
```

3.3.2.2 Description of the move_robot.py code

The script move_robot.py is used to communicate with the robotic arm. The arm is controlled to perform the operations of movement and picking of the tomato.



```python
import numpy as np


def move_robot(socket, old_position, **kwargs):

    bit16rotation = np.array([old_position[1,0], old_position[0,0]], dtype=np.uint8)
    bit16vertical = np.array([old_position[3,0], old_position[2,0]], dtype=np.uint8)
    bit16horizontal = np.array([old_position[5,0], old_position[4,0]], dtype=np.uint8)
    gripper = np.array([old_position[6,0]], dtype=np.uint8)
    boolean_value = True
    reached_rotation = bit16rotation.view(dtype=np.uint16)
    reached_vertical = bit16vertical.view(dtype=np.uint16)
    reached_horizontal = bit16horizontal.view(dtype=np.uint16)

    rotation = bit16rotation.view(dtype=np.uint16)
    vertical = bit16vertical.view(dtype=np.uint16)
    horizontal = bit16horizontal.view(dtype=np.uint16)

    for key, value in kwargs.items():
        match key.lower():
            case 'rotation':
                temp_rotation = value
                rotation = round((temp_rotation+5)*9.85)
                if rotation > 3300:
                    rotation = 3300
                if rotation < 0:
                    rotation = 0
                rotation = np.array([rotation], dtype=np.uint16)
                bit16rotation= rotation.view(dtype=np.uint8)
            case 'horizontal':
                temp_horizontal = value
                horizontal = round(temp_horizontal*821)
                if horizontal > 78:
                    horizontal = 78
                if horizontal < 0:
                    horizontal = 0
                horizontal = np.array([horizontal], dtype=np.uint16)
                bit16horizontal= horizontal.view(dtype=np.uint8)
            case 'vertical':
                temp_vertical = value
                vertical = round(temp_vertical*(-12857))
                if vertical > 1800:
                    vertical = 1800
                if vertical < 0:
                    vertical = 0
                vertical = np.array([vertical], dtype=np.uint16)
                bit16vertical= vertical.view(dtype=np.uint8)
```



```
            case 'gripper':
                gripper = value
    print("GRIPPER: ", gripper)
    codified_var = np.array([[bit16rotation[1]], [bit16rotation[0]],
                             [bit16vertical[1]], [bit16vertical[0]],
                             [bit16horizontal[1]], [bit16horizontal[0]],
                             [gripper],
                             [0]], dtype=np.uint8).tobytes()
    if gripper == old_position[6,0] and np.array_equal(reached_horizontal, horizontal)
        return old_position


    print(codified_var)
    print("codified var stampato")
    socket.sendall(codified_var)
    print("qui debug")
    old_position = np.frombuffer(socket.recv(8), dtype = np.uint8).reshape(-1,1)
    print("qui arrivo? probabilmente no")
    return old_position
```

In this code, the function move_robot() sends commands to the robotic arm to move it to the specified coordinates.

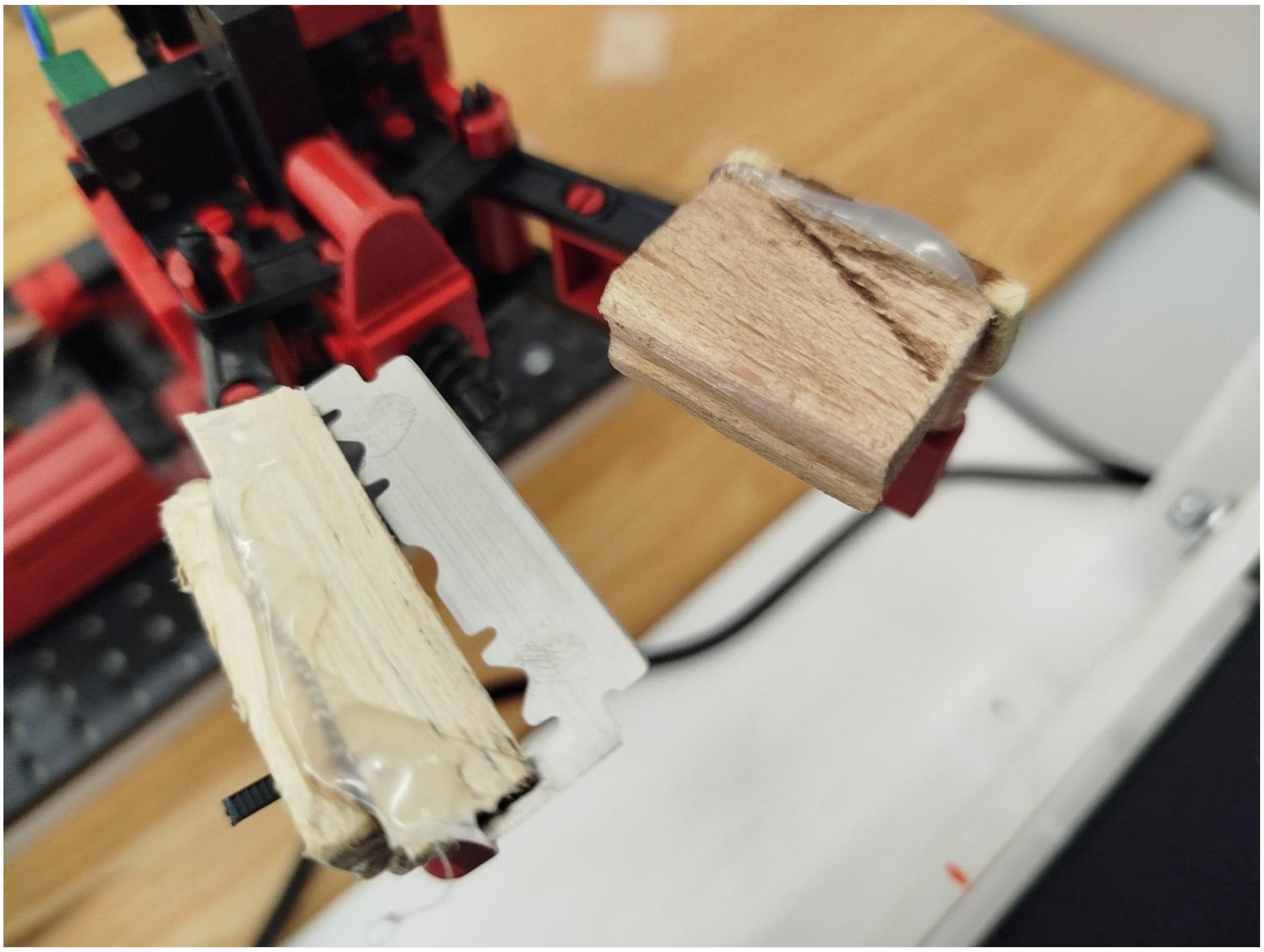
3.3.2.3 TCP/IP communication

The robotic arm and the computer vision system communicate with the Siemens PLC using the TCP/IP protocol. This enables synchronized management of harvesting operations and control of greenhouse environmental parameters. The TCP/IP connection is implemented to ensure that all system.

components work harmoniously, responding promptly to changes detected by sensors and decisions made by the machine learning algorithm for arm control.

4 Cutter Design

To complete the tomato harvesting operation, it was decided to implement a homemade cutter to handle the cutting function. In fact, in a real scenario, it would be necessary to separate the tomato from the plant stem.



The end-effector's grip alone, without the aid of a cutting function, would not allow the tomato to be harvested without causing considerable damage to the plant, and the tomato itself could fall off the conveyor belt.

The Fischertechnik robotic arm, however, is not equipped with a module for common scissors insertion. Furthermore, the plastic structure is not suitable for supporting very heavy tools as they would cause bending of the axes, leading to excessive stress on the joints, which are not designed for such loads beyond their specifications.

Therefore, it was decided to minimize both the space required for introducing the cutter and the weight it would introduce by using few structural elements and reducing to a minimum what is required for cutting. Thus, a blade-beater system was employed, consisting of a razor blade and a wooden beater.

The razor blade takes up little space and is sharp enough to incise and cut the stem holding the tomato. The beater allows the blade to reach the end stop (actually "fictitious," as the pressure applied on the tomato to grab it is determined by a counter in the ladder diagram, as seen previously), providing the necessary force for the operation.

Since the gripper cannot completely close due to the presence of the target inside, the beater is designed to be closer to the blade, so even if the gripper is not fully closed (due to the presence of the target), the blade-beater assembly is already closed at the top, allowing the cutting to occur.

To create the cutter, recycled materials such as plywood and particle board, along with hot glue, were used. Four blocks were cut to the following dimensions:

- Length corresponding to the free space at the end of the gripper
- Width corresponding to the free space of the gripper (its width)
- Height approximately 5 mm higher than the measured height between the gripper and the target. This ensures that the cut is made above the tomato itself, avoiding cutting through it and damaging the machinery.

For each gripper (left and right), two stacked blocks were used, glued together along the length side, with a hole in the middle for the passage of an industrial cable tie commonly used in electrical work. These ties allow the wooden blocks to be positioned and secured to the individual grippers. Using ties was chosen to avoid permanently damaging the lab equipment with adhesive materials.

Finally, the blade was placed on top of the glued blocks, initially positioned in the center after making a notch (to provide stability), and then fixed with hot glue. In the opposite gripper, instead of the blade, the beater was placed. This was another wooden block cut to size to reach the blade at the gripper's minimum opening when in the presence of the tomato.

An additional precaution was taken with the beater by making a central wedge cut, allowing the blade to move towards the center and avoid dangerous bends. A saw for wood, a knife for making cuts, and sandpaper for smoothing corners and creating the wedge were used for designing and crafting the wooden blocks.

5 Conclusions

5.1 Reached Goals

The development of the smart greenhouse has been a good example of how the different technologies are able to interact together and work in cooperation through different layers of logic. This project gave us an insight into a lot of different fields and applications, covering starting to the PLC logic, that set itself more towards the low-level, up to the machine learning, notorious to be a very high-level algorithm.

The goals, intended in the design phase of the project have been reached.

5.2 Encountered Issues

The team encountered few issues during the realization and execution, as the following:

- The gripper of the Fischertechnik robot has proven to be vulnerable to sustained execution of the program, as the continuous movements have damaged parts of its gears until it wasn't able anymore to open and close properly.
  This is another example of how in the industrial environment the quality of the machinery must be high and most importantly with constant maintenance. This factor has to be taken care of when doing the simulation phase of the design.

- To obtain a work environment simulation that is as truthful and reliable as possible, and given the limitations imposed by Siemens software 'PLCSIM S7', the "PLCSIM ADVANCED" software was employed to expand the simulation environment to include communication between the PLC and the computer.

- As mentioned before, the PLC available in the laboratory didn't have enough ports to achieve the complete execution of the whole program, including greenhouse management and robotic arm duty both working at the same time. Another point to notice is that even with just the robotic arm connected, there were not enough ports to link all the lights, as originally intended in the first version of the project

- The wide availability of functions provided by the TIA Portal program makes it a very powerful tool, but with a rather steep learning curve, which initially made working on this platform challenging but stimulating.

5.3 Future Goals

Based on the work conducted in this paper, it is possible to outline potential future developments.

Firstly, the use of a different model of PLC with an adequate number of I/O ports would have made it possible to implement both sections of the system, allowing for the initial simulation and subsequent physical realization of the entire apparatus.

Secondly, the management and conditioning of the sensors used to maintain optimal conditions inside the greenhouse, which in this paper have been only superficially addressed, offer an implementation possibility using Arduino and its suite.

Finally, in order to make plant harvesting quicker and more efficient, a design phase hypothesized a system that includes the presence of a second position in front of the existing one. This, in addition to improving harvest performance, resolves one of the potential issues of the current project, namely the possibility of a fruit being located behind one of the plant's branches.

6. Sitography

1) https://github.com/vincenzobucaria/smart-greenhouse
2) https://github.com/enrikata/IAAR-Project/tree/main