Faculty of Engineering and IT

Subject: 48560 Automation Studio

Assignment Number: Assignment 4 **Date Submitted: 17 Nov 2025**

Assignment Title: Final Project Report

Student Name(s) and Number(s) **Tutorial Group: 01**
Matthew Chua 24837546
Marcus Frischknecht 24428068

Signature:
Marcus Frischknecht
Matthew Chua

Statement of Collaboration:

We worked together collaboratively. Marcus worked on the reverse engineering of the station, and both members worked on the UR3, OpenCV, Python script, Raspberry Pi, and PLC logic.

UNIVERSITY OF TECHNOLOGY SYDNEY

Demonstration Practice 3 – Local and Remote Monitoring

45860 Automation Studio
Spring 2025
Coordinator: A/Prof Ricardo P. Aguilera

Student Name	Matthew Chua
Student Number	24837546
Contribution	80%

Student Name	Marcus Frischknecht
Student Number	24428068
Contribution	100%

Vision-Based Quality Control and Correction for the SIF-408 Palletizing Station

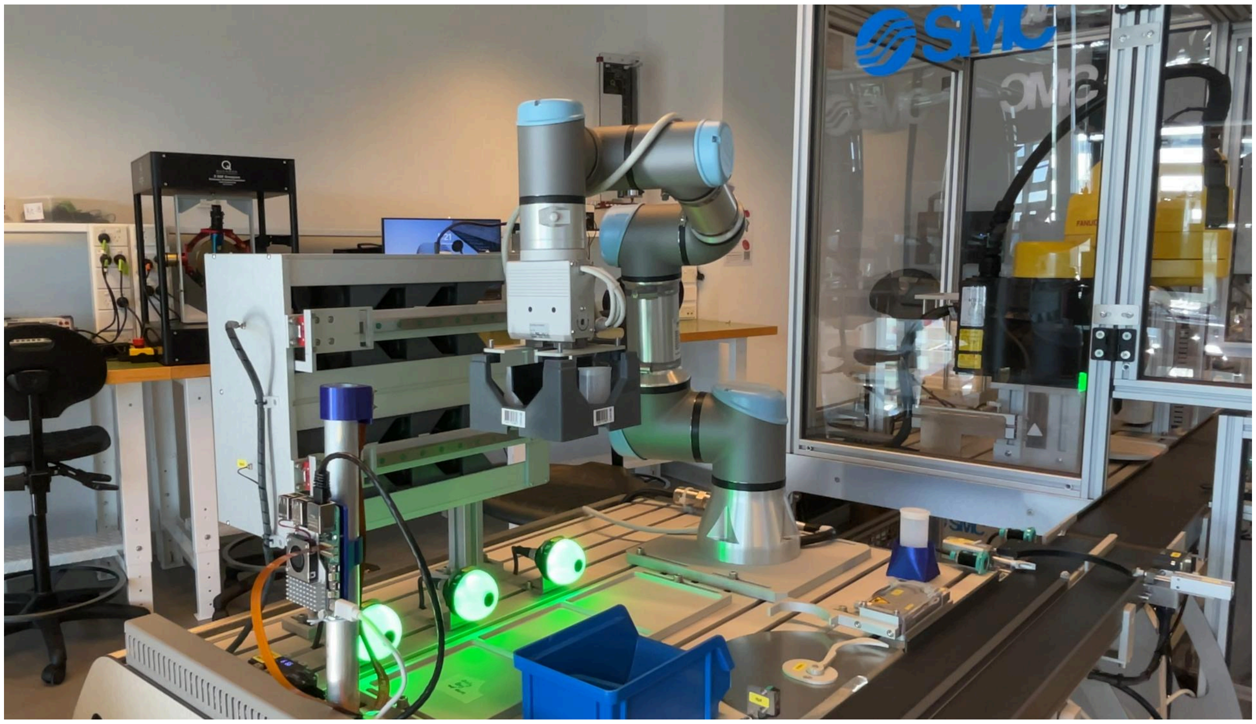


Figure 1: Modified SIF408 station with UR3 positioning pack in front of camera for quality inspection

CLICK: Vision (RPI) & ROBOT (UR3) CODE REPOSITORY CLICK: PLC PROJECT FILES

Contents

Abstract

This project extends the functionality of the SIF-408 palletizing station by implementing a vision-based quality-control subsystem to improve the reliability of cylindrical container placement. The base system occasionally places containers at a tilted angle, creating downstream faults in later stations such as SIF-410. To address this issue, a Raspberry Pi with a side-mounted camera was integrated into the existing workflow, and custom software using OpenCV was developed to detect misaligned containers. The UR3 robot was then modified to present the pack for inspection and perform corrective pushing actions when required.

The PLC and UR3 programs were reverse-engineered, analysed with assistance from generative AI, and modified to support a new quality-control loop without altering the station's existing Profinet configuration. The system successfully detects misalignment, triggers corrective actions, and verifies corrections before allowing the pack to proceed. Testing demonstrated reliable detection and correction of physical misalignment, significantly improving station robustness at the cost of a small increase in cycle time. This work demonstrates a practical and modular approach to integrating closed-loop quality control into an existing Industry 4.0 training system.

1. Introduction

The SIF-400 Industry 4.0 training system is a modular, flexible educational platform that simulates a highly automated smart factory. This smart factory is designed to produce two types of containers (cylindrical and squares) with three different content types (solids, liquids, customised). The entire process is distributed across a series of up to 14 modular stations which are grouped into either assembly and production tasks, logistic tasks, or auxiliary tasks.

Stations SIF-401 through SIF-407 handle tasks such as filling containers, capping and labelling which falls under the “assembly and production” category. Stations SIF408-SIF-412 manage packaging, warehousing, palletising and dispatch which falls under the “logistics” category. Finally, stations SIF-413 and SIF-414 fall under the “auxiliary” umbrella.

This report will focus on the critical link station, SIF-408, bridging the production and logistics processes. This station is of particular interest as it introduces collaborative robotics; a technology specifically designed for human-robot interaction into the automated workflow, making the station align closest to the team’s mechatronics expertise.

1.1 Base Functionality

The purpose of the SIF-408 container packing station is to group finished containers into packs to prepare for shipment or warehouse storage. Utilising the UR3 collaborative robot, the SIF-408 organises packs according to the user specification via the SIFMES-400 Management System.

The standard operational workflow for the SIF-408 station begins when a container arrives on its transport pallet via the modular transfer belt. At the station's entrance, an RFID reader identifies the pallet to determine which production order the container belongs to. Simultaneously, the UR3 collaborative robot retrieves an empty pack from the station's horizontal warehouse and places it onto one of the three available packing posts. These posts are equipped with capacitive light sensors that indicate their status: **green** (free), **yellow** (in progress), or **red** (complete). The cobot then picks the container from its entry position and places it into the pack. Immediately after, the robot removes the now-empty pallet from the entry position and drops it into a collection drawer. This process repeats until the pack is filled with the specified number of containers (up to four) as defined by the SIFMES-400 management system's order. Once the pack is complete, the robot moves the finished pack from the post onto the transfer belt, sending it to the next station in the logistics line.

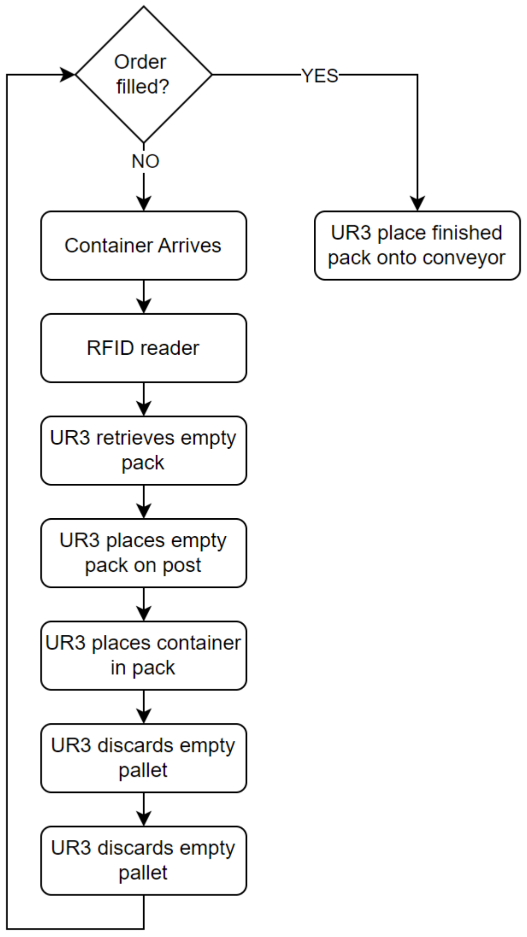

Order filled?
YES
NO
Container Arrives
UR3 place finished pack onto conveyor
RFID reader
UR3 retrieves empty pack
UR3 places empty pack on post
UR3 places container in pack
UR3 discards empty pallet
UR3 discards empty pallet

Figure 2: SIF408 process flowchart

1.2 Key Issues with Base Functionality

During various demonstrations of the SIF-400 system, it was noted that the UR3 cobot would occasionally fail to place cylindrical containers perfectly upright within the packs. The containers would be successfully placed inside the pack but would be tilted. This misalignment was observed to cause issues in later stages of production, such as at the SIF-410 palletizing station.

On rare occasions, a tilted container could fall over completely during transit on the transfer belt. This raises significant issues regarding quality control and introduces unexpected disruptions and potential line stoppages in later processing, suggesting the need for a robust quality check before allowing packs to leave the SIF-408 station.

1.3 Problem Statement

The placement of cylindrical containers by the SIF-408's UR3 cobot lacks sufficient reliability, resulting in misaligned (tilted) containers within the final pack. These misalignments compromise the integrity of the pack and can lead to failures in downstream automated processes (e.g., palletizing, wrapping, and dispatch).

The base system lacks a dedicated quality control step to verify container alignment after placement and before shipment. This project aims to solve this problem by integrating an automated vision inspection and correction system to ensure all cylindrical containers are properly seated before the pack is approved to move to the next station.

1.4 Objectives

The primary objective of this project is to design, implement, and integrate a vision-based quality control system into the SIF-408 station to improve the alignment reliability of cylindrical containers.

Secondary objectives include:

- To research and apply computer vision techniques using OpenCV for industrial inspection.
- To gain experience with Industry 4.0 communication protocols by interfacing a Raspberry Pi with an industrial PLC and a UR collaborative robot.
- To develop a non-invasive corrective action using the existing UR3 robot, avoiding the need for additional mechanical actuators.

1.5 Methods

To achieve these objectives, the following methodology was employed:

1. **Reverse Engineering:** The team first analysed the existing SIF-408 station, including its PLC logic, the UR3's PolyScope program, and the communication protocols (Profinet/OPC UA) used to coordinate the station's components including the gripper.
2. **System Design:** A new quality control subsystem was designed, consisting of a Raspberry Pi, a side-mounted camera, and custom software. This system was designed to be injected into the existing process flow.
3. **Prototyping:** The vision logic was developed in Python using OpenCV. Communication links between the Raspberry Pi, PLC, and UR3 were established via Modbus TCP.
4. **Integration:** Custom logic was added to the UR3's program to present the pack to the camera for inspection and to perform a corrective "push" action. The PLC logic was modified to accommodate this new quality control state in the station's workflow.
5. **Validation:** The complete system was tested to validate its ability to correctly detect and rectify misaligned containers.

2. SIF System Architecture

The SIF-400 system employs a modular and decentralized architecture at the station level, managed by a centralized supervisory system, SIFMES-400, for high-level coordination and order processing. Each of the 14 individual stations operates as an independent module, controlled by its own dedicated SIEMENS Programmable Logic Controller (PLC), from the SIMATIC S7-1500 series. These PLCs are the "brains" of their respective stations, responsible for all real-time control tasks, including reading sensor data and actuating components like motors, pneumatics, and robots. All programming and logic for these controllers are developed within the SIEMENS TIA (Totally Integrated Automation) Portal, which provides a unified engineering environment.

For broader system integration, the station PLCs utilize industrial Ethernet protocols. The primary protocol in this SIEMENS ecosystem is PROFINET, which is used for deterministic, high-speed communication with intelligent field devices such as RFID readers, robot controllers (like the UR3), and motor drivers. This robust communication is also used for the distributed I/O modules controlling the transfer system.

Overarching control and coordination are provided by the SIFMES-400 management software, which acts as the Manufacturing Execution System (MES) and SCADA layer. SIFMES-400 handles order management, production tracking, data logging, and overall system supervision. All the SIEMENS PLCs communicate with the central SIFMES-400 server via an Ethernet network using the OPC UA (Open Platform Communications Unified Architecture) protocol. Modern SIEMENS controllers have integrated OPC UA server capabilities, making this a seamless and standard method for secure data exchange. This ensures reliable communication between the distributed PLCs and the central management system, enabling the seamless coordination of the entire smart factory workflow.

3. SIF-408 System Architecture

3.1 Research Methodology

Upon starting the project, we retrieved the PLC program from the controller and began to reverse engineer the program blocks. “Sif408_steps” was considered to be the only program-block file of interest to us, as it contained the state-machine controlling the station's output and input device logic. With over 50 states within the state-machine, few comments and no intrinsic documentation, it became very difficult to manually reverse engineer the system and understand how it works. To better understand the logic running on the UR3 and the communication methods used between the robot and PLC, we copied the .script file from the UR3’s teach pendant and also began to reverse engineer how the robot worked. We learned that it also consisted of a large state-machine that was influenced by the PLC.

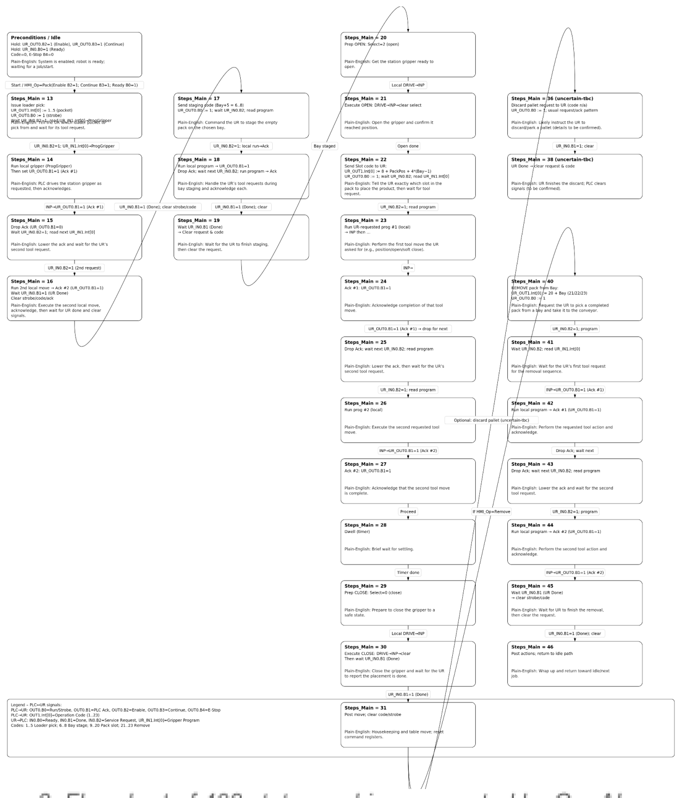


Figure 3: Flowchart of 408 state machine generated by GenAI reverse engineering structured text

Following advice from the coordinator, GenAI was used for time-efficient reverse-engineering of our SIF408 station code. PDF printouts of the PLC step code and the .script file were uploaded to GPT5 along with carefully crafted prompts asking the LLM to analyse both files and summarise how the system worked. We also prompted it to generate SVG flowcharts (see figure 3) visually demonstrating the flow of different states and CSV printouts of the different states and what values IO variables would likely be for each state with supporting descriptions. With these resources we quickly were able to understand how the two state-machines were influencing each other and validated the LLM’s output by ‘live-monitoring’ the PLC and manually jogging the robot through its state machine and subfunctions. Once we were familiar with how the station was operating, we could identify where the state machines required modification to implement our quality control code.

3.2 UR3

Analysis of the UR3 .script code revealed the UR3 also operates as a state machine. The PLC communicates to the UR3 what action it should perform (e.g pick pack from charger, place container in pack, unload pack and which position for each action e.t.c). This is done through the switch case structure based on the 'InNET_Operation' variable. Each action consists of a set of functions executing through a series of pre-programmed waypoints. Interestingly, it was observed the gripper attached to the robot is controlled through the PLC. The UR3 simply requests what position it would like the gripper to open to and then waits in a blocking loop for the PLC to send boolean confirmation the gripper is in position.

Initially, when reverse engineering the UR3's communication with the PLC, we expected it to be using OPC-UA but were surprised to learn it was actually using profinet. This discovery however did not impact our project as we were able to use the existing PLC tags rather than modifying communication interfaces.

3.3 PLC

The PLC code operating on the station is a state-machine of more than 50 states programmed in Structured text. The code contained almost no user comments and variable names were ambiguous. Reverse engineering the PLC code was challenging and reliance on the resources produced by an LLM were crucial to assist with understanding the inner workings of the system within a short timeframe. States 40-45 became the primary focus within our investigation. Within state 40, the PLC communicates to the UR3 which pack is to be unloaded from the station and the subsequent states were responsible for awaiting gripper requests, forwarding requests to the gripper and sending gripper position acknowledgement booleans back to the robot. This section of the state-machine was fragile as it built a strong 'handshake and acknowledgement' mechanism between the robot, PLC and gripper. A 'time-out' fault detection mechanism seemed to be included in the logic but we never saw it activate an alarm or error during our times of testing and modification.

3.4 Gripper (SMC LEC LEHF20K2-24-R3CP18)

The gripper operates using a set of predefined opening positions that are requested by the PLC. These positions are configured using SMC's proprietary controller-setting software rather than being defined within the PLC program itself. Communication between the PLC and the gripper is handled via a Profinet interface, allowing the PLC to command position changes and receive status information directly from the gripper controller.

4. SIF-408 Canister Levelling System Architecture

Our solution is a modular subsystem designed to integrate seamlessly into the existing SIF-408 workflow. It intercepts the process immediately after the robot has been tasked to pick a pack and place it on the exit conveyor.

4.1 Overview

As shown in the diagram below, the canister levelling subsystem is injected into the base functionality of the system and runs after an order has been completed by the SIF-408.

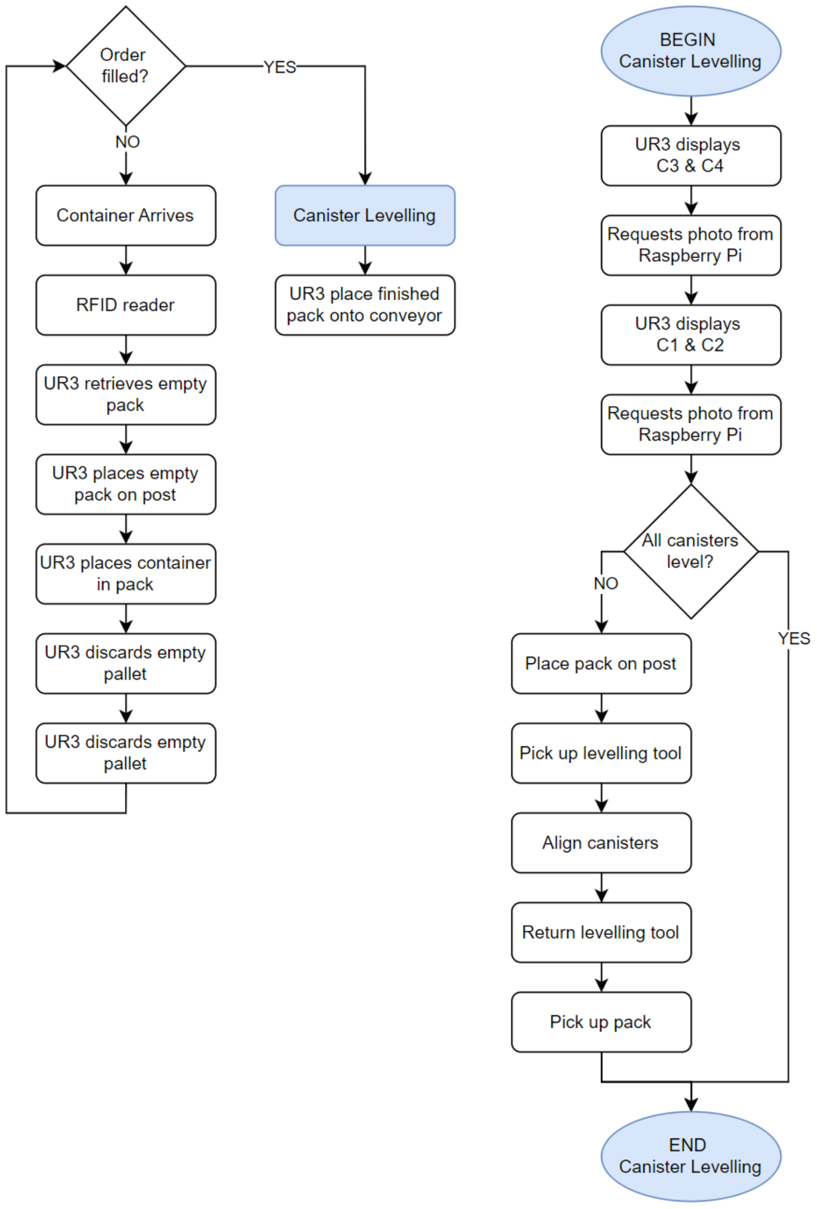


Figure 4: Flowchart of container levelling process

The new "Canister Levelling System" injects a quality control loop into the base functionality. The high-level process flow is as follows:

1. The UR3 robot completes its standard task of filling a pack with cylindrical containers.
2. Instead of immediately moving the pack to the transfer belt, when the PLC logic ("steps_main") enters case 43, the case loops continually with modified code allowing the UR3 to request arbitrary gripper positions from the PLC until the inspection and pack exit-conveyor loading process is completed.
3. The UR3 picks up the completed pack and rotates it to present a side-on view to a newly installed Raspberry Pi camera. It first displays canisters C3 and C4, and then turns the pack 180 degrees, displaying canisters C1 and C2.
4. The Raspberry Pi captures an image, processes it using an OpenCV script to detect misaligned (tilted) containers, and sends a "Pass" or "Fail" boolean for each of the canisters back to the PLC/UR3 via Modbus TCP.
5. **If all canisters "Pass":** The UR3 proceeds to its normal "place_on_conveyor" routine.
6. **If any canisters "Fail":** The UR3 places the pack back on the packing post and executes a new "recorrection" subroutine, firmly pushing down on the misaligned containers to reseat them.
7. The UR3 then proceeds to its normal "place_on_conveyor" routine.

See Figure 5 to understand the communication interfaces used between different devices within the SIF408 station.

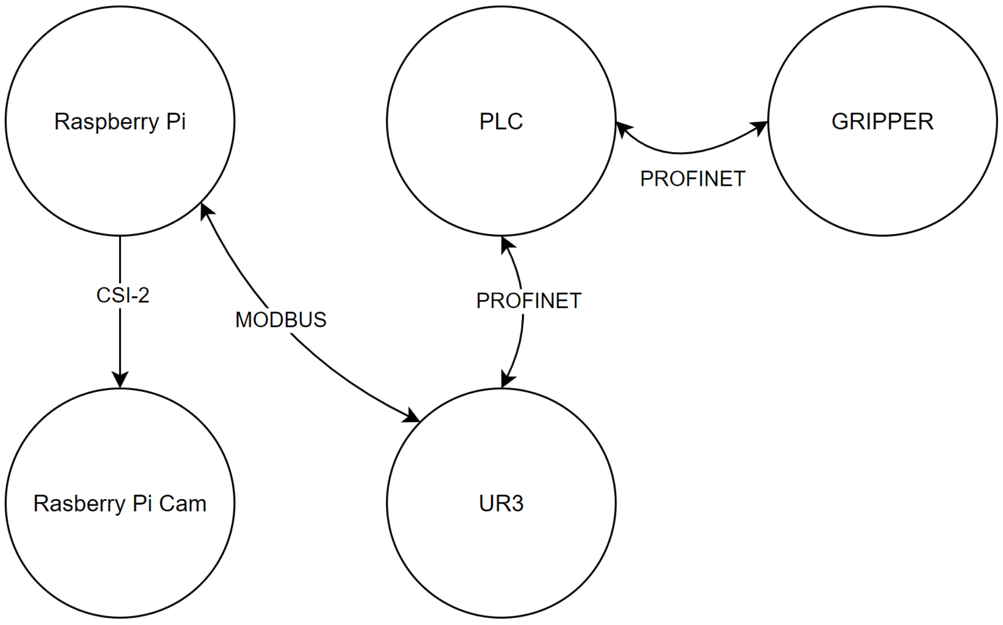


Figure 5: Communication interface diagram

4.2 PLC

The station's PLC logic was modified to accommodate the new quality control state, specifically in the SIF308_STEPS [FC1] function. State 43 runs once a pack unload has been requested, and code has been injected into this state, replacing the original functionality. Instead of transitioning directly to placing the pack on the conveyer the PLC now waits until the UR3 confirms the packs are level through setting the "Robot_UT_INT1"."INT"[0] register to the value of 8 before incrementing "Steps_Main" to 44 (the next value). During this wait in step 43 the PLC accepts any gripper requests from the UR3. This enables the UR3 to interact with the pack and custom tool via the gripper during the levelling stage and allows the functionality of canister levelling to be achieved without additional cases, potentially leading to confusing case numbers in the PLC code.

No extra tags or Profinet integration was utilised to achieve the levelling subsystem. Instead, the UR3 triggers custom states via existing tags. Any functionality that could be integrated with new states was implemented using redundant registers during step 43. Though this isn't standard industry practice, no documentation was provided which linked the registers of the UR3 to the PLC registers. As such, the function of applicable registers were discovered via reverse engineering the system. Since there was no documentation to aid the creation of new tags, it was decided that the existing gripper states and registers would be used to communicate between the UR3 and PLC.

4.3 UR3

Before the UR3's sequence to place the pack onto the conveyor, the levelling code is injected. The UR3 picks up the pack and positions it in front of the camera. Via ModBus, it sends a signal to the Pi, which requests a photo to be taken. Once the Pi confirms the photo has been taken and it's ready to take the second, it sends a message to the UR3 via ModBus which subsequently twists the pack 180 degrees to display the next set of items. Again, it sends a signal to the Pi to take the second photo. Upon receiving confirmation from the Pi that the final photo has been taken, the UR3 waits 2 seconds. During this time, the Pi processes its images, determining if any of the canisters require realignment. In the case that all items in the pack are satisfactory, then the UR3 moves the pack onto the conveyor. However, if there is at least one canister requiring realignment, the UR3 places it back in the bay and fixes the canisters using a tool to push them down again. When the corrective action has been executed, the UR3 puts away the tool, and places the pack onto the conveyor.

4.4 Gripper (LEHF20K2-24-R3CP18)

Initially, the gripper was supposed to squeeze its prongs shut together and align canisters without the need for a tool. However, it was discovered that the mechanical design of the gripper meant the prongs could not close together. The gripper is set up to move to these distinct positions:
Due to the limitation on closing the prongs, the decision was made to use a spare cylindrical canister as a tool for the gripper to perform its realignment. (See figure 6)

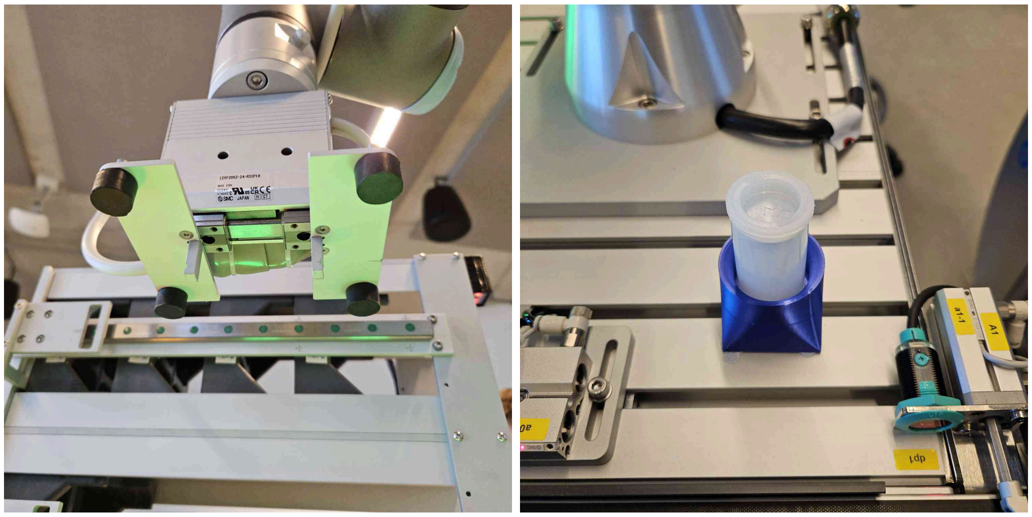


Figure 6: SIF408 gripper underside (Left)
Toolbit used for correction stored in holder (Right)

4.5 Raspberry Pi Camera

The Raspberry Pi Camera Module 3 is used for image acquisition. It is mounted on a custom 3D-printed bracket, positioning it to have a clear, side-on view of the containers as the UR3 presents them (See figure 7). A side view was chosen as it provides superior depth cues for assessing if a cylindrical container is level, a determination that is very difficult from a top-down perspective. The camera was specifically positioned at a height that enabled the twisting of packs without the risk of collisions with other equipment. Communication with the Raspberry Pi is handled via the CSI-2 (Camera Serial Interface 2) protocol.

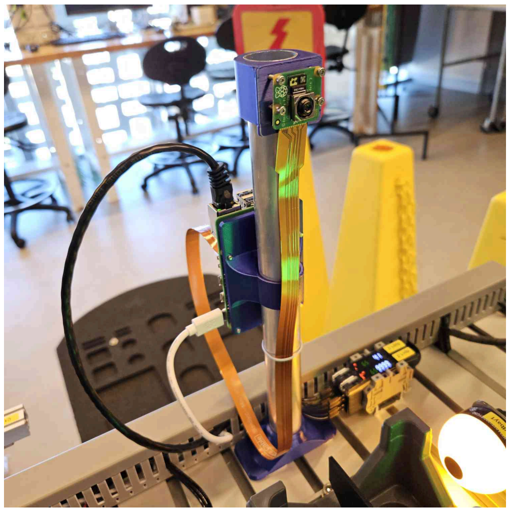


Figure 7: Quality inspection camera mounted on pole

4.6 Raspberry Pi

A Raspberry Pi 5 acts as the brain of our quality control system. It is mounted on a custom 3D-printed bracket which is secured on an aluminum tube to the side of the station's bench (See figure 8). This ensured we could run our power and ethernet cables through the stock cable racks, and that the extra hardware would be out of the way of any other operations. The Pi acts as a Modbus TCP server, interfacing with the UR3 controller to receive the PHOTO_READY_STEP_ADDR trigger and send back the “recorrect” booleans. The server also logs every inspection result (Pass/Fail) to the command line, including general debugging and runtime information such as ModBus connection and key steps that are being run such as photo capture.

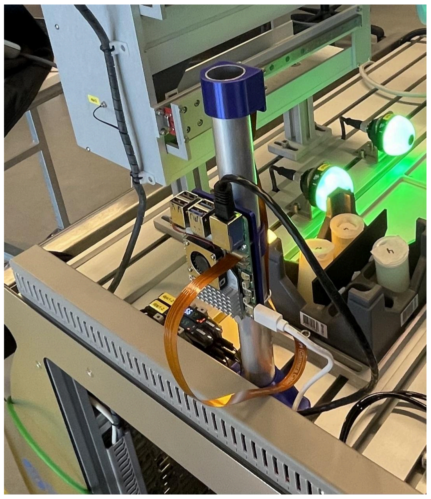


Figure 8: Raspberry Pi mounted on pole

A Python script utilizing the OpenCV library runs on the Pi. Upon startup, the Pi starts a server which the UR3 connects to via ModBus. When the UR3 requests image capture via the ModBus flag PHOTO_READY_STEP_ADDR, the Pi captures the first image of C3 and C4. Then, after the UR3 spins the pack, it requests the second picture and the Pi captures the second image of C1 and C2. Following this, it processes the photo including cropping it, and turning it black and white, before performing edge detection using OpenCV’s Canny and Hough functions to identify the top edge of the container. Each container’s tilt angle is then calculated, and if any container's tilt exceeds an angle of 1.9 degrees, or has a curvature of more than 4, the script determines the container requires re-correction. A basic algorithm proved sufficient (see figures 9 & 10), eliminating the need for a complex AI model and opening the possibility for future work to port this logic to a simpler microcontroller like an ESP32.

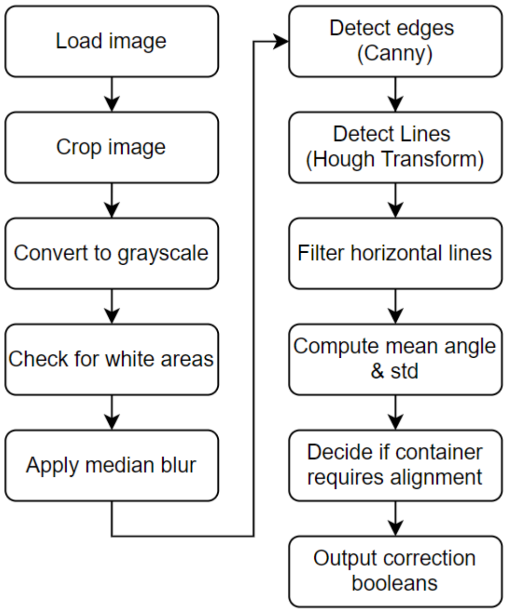


Figure 9: Flowchart of image processing steps

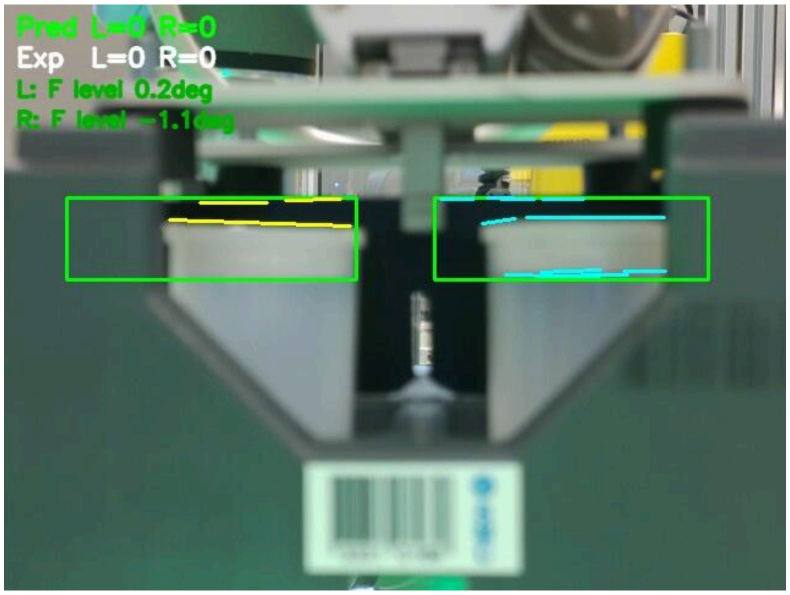


Figure 10: Debug image output from image processing demonstrating edge detection for lid analysis

5. Results

5.1 Project Results

The integrated Canister Levelling System was successfully implemented and validated. The system reliably identifies and corrects misaligned cylindrical containers, demonstrably improving the output quality of the SIF-408 station.

In operation, the modified workflow functions as designed:

1. Once the pack is filled, the UR3 robot correctly picks up the pack and rotates it in front of the side-mounted camera.
2. The Raspberry Pi successfully captures and processes the image, with the OpenCV script accurately identifying tilted containers.
3. Upon receiving a “recorrect” boolean from the Pi (via Modbus TCP), the UR3 robot places the pack back and executes the our custom correction subroutine, firmly pushing the misaligned container(s) until level.
4. If the pack passes inspection (either on the first try or after correction), the robot moves it to the transfer belt to continue to the next station.

5.2 Raspberry Pi configuration Documentation

Network Settings:

- Static IP: 130.130.130.202
- Subnet Mask: 255.255.255.0

- Hostname: `408pi` (changed from the default “raspberrypi” to avoid conflicts with other devices)
- Connected via Ethernet to one of the SIF408 station network switches

To modify network settings, use:
sudo nmtui

Access Credentials:

- Username: admin
- Password: admin

SSH access:

```
ssh admin@408pi.local
ssh admin@130.130.130.202
```

Internet Access Notes:
The SIF400 network does not provide internet access.
If temporary access is needed, SSH into the Pi and use `sudo nmtui` to connect it to a phone hotspot.

Project Repository:
All project files are stored in:
~/sif408_camera

Repository contents:

- vision_main.py – Main script performing automatic quality control checks.
- lid_detection.py – Helper script for lid/canister angle detection.
- save_photos_for_calibration.py – Utility for capturing calibration images.
- simulate_main.py – Used for simulation and offline testing.

Repo link: https://github.com/marcus-frisch/sif408_camera

Process Management (PM2):
PM2 is used to automatically run and supervise the vision_main.py script.

Purpose of PM2:

- Starts vision_main.py automatically on boot.
- Restarts the script if it crashes.
- Provides tools for checking logs and controlling the process.

Important Notes:

- The Python scripts function as a Modbus TCP server for the UR3.
- The server runs on port 502, which requires root privileges.
- Only one Modbus server can run at a time, so only one script can be active.
- When running scripts manually, use: sudo python

Using PM2:

All PM2 commands must use sudo. Running PM2 without sudo creates a separate, incorrect environment.

Check status:

```
sudo pm2 status
```

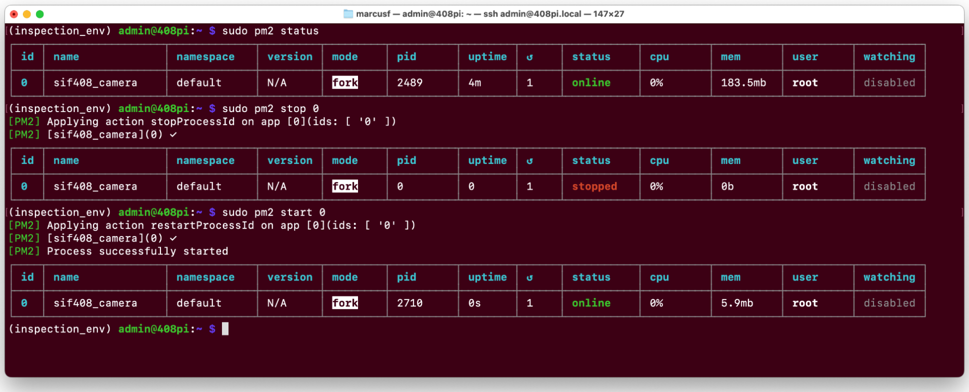
You should see a process named sif408_camera (ID 0), which corresponds to vision_main.py (see figure 11).

Stop or restart the script:

```
sudo pm2 stop 0
sudo pm2 start 0
```

Alternatively:

```
sudo pm2 stop sif408_camera
sudo pm2 start sif408_camera
```

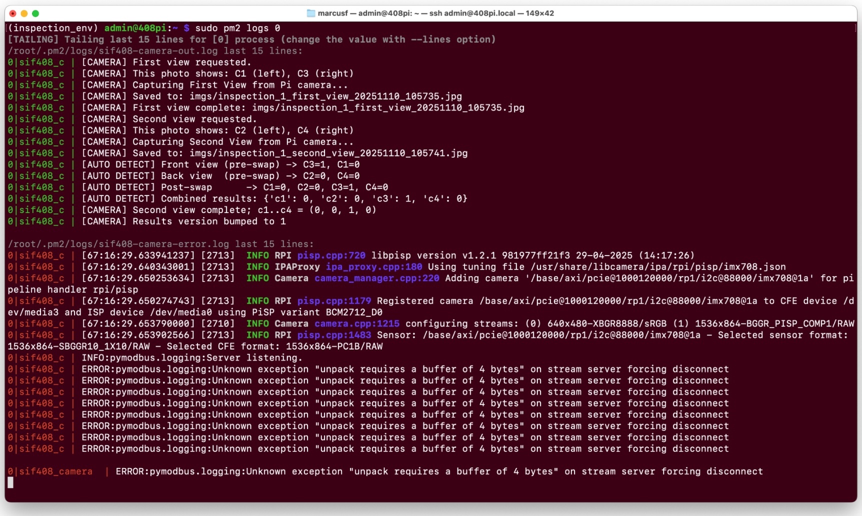


```
marcusf — admin@408pi: ~ — ssh admin@408pi.local — 147×27
(inspection_env) admin@408pi:~ $ sudo pm2 status
│ id │ name          │ namespace │ version │ mode │ pid  │ uptime │ ↺ │ status  │ cpu │ mem     │ user │ watching │
│ 0  │ sif408_camera │ default   │ N/A     │ fork │ 2489 │ 4m     │ 1 │ online  │ 0%  │ 183.5mb │ root │ disabled │
(inspection_env) admin@408pi:~ $ sudo pm2 stop 0
[PM2] Applying action stopProcessId on app [0](ids: [ '0' ])
[PM2] [sif408_camera](0) ✓
│ id │ name          │ namespace │ version │ mode │ pid  │ uptime │ ↺ │ status  │ cpu │ mem     │ user │ watching │
│ 0  │ sif408_camera │ default   │ N/A     │ fork │ 0    │ 0      │ 1 │ stopped │ 0%  │ 0b      │ root │ disabled │
(inspection_env) admin@408pi:~ $ sudo pm2 start 0
[PM2] Applying action restartProcessId on app [0](ids: [ '0' ])
[PM2] [sif408_camera](0) ✓
[PM2] Process successfully started
│ id │ name          │ namespace │ version │ mode │ pid  │ uptime │ ↺ │ status  │ cpu │ mem     │ user │ watching │
│ 0  │ sif408_camera │ default   │ N/A     │ fork │ 2710 │ 0s     │ 1 │ online  │ 0%  │ 5.9mb   │ root │ disabled │
(inspection_env) admin@408pi:~ $ 
```

Figure 11: Checking process status, starting and stopping a process in PM2

View logs (see figure 12):

```
sudo pm2 logs 0
```



```
marcusf — admin@408pi: ~ — ssh admin@408pi.local — 149×42
(inspection_env) admin@408pi:~ $ sudo pm2 logs 0
[TAILING] Tailing last 15 lines for [0] process (change the value with --lines option)
/root/.pm2/logs/sif408-camera-out.log last 15 lines:
0|sif408_c | [CAMERA] First view requested.
0|sif408_c | [CAMERA] This photo shows: C1 (left), C3 (right)
0|sif408_c | [CAMERA] Capturing First View from Pi camera...
0|sif408_c | [CAMERA] Saved to: imgs/inspection_1_first_view_20251110_105735.jpg
0|sif408_c | [CAMERA] First view complete: imgs/inspection_1_first_view_20251110_105735.jpg
0|sif408_c | [CAMERA] Second view requested.
0|sif408_c | [CAMERA] This photo shows: C2 (left), C4 (right)
0|sif408_c | [CAMERA] Capturing Second View from Pi camera...
0|sif408_c | [CAMERA] Saved to: imgs/inspection_1_second_view_20251110_105741.jpg
0|sif408_c | [AUTO DETECT] Front view (pre-swap) -> C3=1, C1=0
0|sif408_c | [AUTO DETECT] Back view  (pre-swap) -> C2=0, C4=0
0|sif408_c | [AUTO DETECT] Post-swap      -> C1=0, C2=0, C3=1, C4=0
0|sif408_c | [AUTO DETECT] Combined results: {'c1': 0, 'c2': 0, 'c3': 1, 'c4': 0}
0|sif408_c | [CAMERA] Second view complete; c1..c4 = (0, 0, 1, 0)
0|sif408_c | [CAMERA] Results version bumped to 1

/root/.pm2/logs/sif408-camera-error.log last 15 lines:
0|sif408_c | [67:16:29.633941237] [2713]  INFO RPI pisp.cpp:720 libpisp version v1.2.1 981977ff21f3 29-04-2025 (14:17:26)
0|sif408_c | [67:16:29.640343001] [2713]  INFO IPAProxy ipa_proxy.cpp:180 Using tuning file /usr/share/libcamera/ipa/rpi/pisp/imx708.json
0|sif408_c | [67:16:29.650253634] [2713]  INFO Camera camera_manager.cpp:220 Adding camera '/base/axi/pcie@1000120000/rp1/i2c@88000/imx708@1a' for pi
peline handler rpi/pisp
0|sif408_c | [67:16:29.650274743] [2713]  INFO RPI pisp.cpp:1179 Registered camera /base/axi/pcie@1000120000/rp1/i2c@88000/imx708@1a to CFE device /d
ev/media3 and ISP device /dev/media0 using PiSP variant BCM2712_D0
0|sif408_c | [67:16:29.653790000] [2710]  INFO Camera camera.cpp:1215 configuring streams: (0) 640x480-XBGR8888/sRGB (1) 1536x864-BGGR_PISP_COMP1/RAW
0|sif408_c | [67:16:29.653902566] [2713]  INFO RPI pisp.cpp:1483 Sensor: /base/axi/pcie@1000120000/rp1/i2c@88000/imx708@1a - Selected sensor format:
1536x864-SBGGR10_1X10/RAW - Selected CFE format: 1536x864-PC1B/RAW
0|sif408_c | INFO:pymodbus.logging:Server listening.
0|sif408_c | ERROR:pymodbus.logging:Unknown exception "unpack requires a buffer of 4 bytes" on stream server forcing disconnect
0|sif408_c | ERROR:pymodbus.logging:Unknown exception "unpack requires a buffer of 4 bytes" on stream server forcing disconnect
0|sif408_c | ERROR:pymodbus.logging:Unknown exception "unpack requires a buffer of 4 bytes" on stream server forcing disconnect
0|sif408_c | ERROR:pymodbus.logging:Unknown exception "unpack requires a buffer of 4 bytes" on stream server forcing disconnect
0|sif408_c | ERROR:pymodbus.logging:Unknown exception "unpack requires a buffer of 4 bytes" on stream server forcing disconnect
0|sif408_c | ERROR:pymodbus.logging:Unknown exception "unpack requires a buffer of 4 bytes" on stream server forcing disconnect
0|sif408_c | ERROR:pymodbus.logging:Unknown exception "unpack requires a buffer of 4 bytes" on stream server forcing disconnect
0|sif408_c | ERROR:pymodbus.logging:Unknown exception "unpack requires a buffer of 4 bytes" on stream server forcing disconnect

0|sif408_camera  | ERROR:pymodbus.logging:Unknown exception "unpack requires a buffer of 4 bytes" on stream server forcing disconnect
```

Figure 12: Checking process output logs in PM2

Running Other Scripts:
To run any other Python script, first stop the PM2-managed version script to avoid port conflicts, then run the desired script with sudo. E.g: `sudo $(which python3) simulate_main.py`

5.3 Project Validation

Validation was performed through repeated visual confirmation of the system's operation.

- **Detection:** We manually created test packs with deliberately misaligned containers. In 100% of these test cases, the vision system correctly identified the "Fail" state.
- **Correction:** In all tests, the system passed as intended. A separate script was run to test the accuracy of the Raspberry Pi detection algorithm. It seems the server only identifies 75-80% of cases successfully over a sample size of 55 samples; many more samples than could be feasibly tested due to the project's timeline.
    - **False Positives:** The system was also validated for false positives. Packs that were visibly correct were inspected and correctly identified as "Pass" in all test runs, preventing unnecessary correction cycles.
    - **False Negatives:** This is the most concerning issue. During physical testing, no false negatives were detected. However, the system theoretically misidentifies 1 in 5 canisters. Further testing needs to be conducted to determine whether these are false positives or negatives.

5.4 Evaluation of Productivity Improvements

A quantitative analysis of the overall productivity improvement is inconclusive without further long-term testing. The project's primary goal was to improve quality and reliability.

The new quality control loop introduces a time-cost to the SIF-408's cycle (approximately 23 seconds for inspection and an additional 50 seconds for a correction cycle). However, this "micro-stoppage" at the SIF-408 station is highly preferable to a full-line stoppage downstream (e.g., if a fallen container jams the SIF-410 palletiser). The project successfully trades a small, predictable increase in cycle time for a large increase in system reliability and a reduction in unpredictable, high-impact downtime.

However, during the research of this project, it was noted that the waypoints and force sensors of the UR3 could be modified in order to theoretically guarantee the optimal placement of the canisters. Automation factories are very precise, and are therefore generally run in an open loop fashion. The addition of functionality to run processes as a closed loop adds time and a high cost to a manufacturing plant. It is recommended instead to improve the waypoints and force sensor thresholds in order to enable the robot's placement of the canisters to be more reliable. Due to the other goals of this project, a camera detection approach was pursued nonetheless.

7 Conclusion

The integration of the vision-based quality control subsystem into the SIF408 station successfully addressed the recurring issue of the misaligned cylindrical containers and improved the overall reliability of the palletizing workflow. This was achieved by reverse engineering the PLC and UR3 programs and augmenting them with a camera inspection loop running on a Raspberry Pi. Testing demonstrated the subsystem successfully detected tilted containers and initiated corrective actions. Although the solution introduces a small increase in cycle time, it significantly reduces the risk of downstream faults and station stoppages, providing a net improvement in system performance. This project showcases a practical approach to retrofitting closed-loop quality control into an existing industry 4.0 training environment and highlights opportunities for future refinement in vision accuracy, robot placement precision and system integration.

7.1 Future Improvements

There are a number of improvements that can be made to the vision based quality-control process introduced to the SIF408 station.

1. Improvement to the camera detection algorithm.
    a. Detection of ghost containers
       Based on testing, the system has a 90% success rate at identifying round containers that require correcting, however it's been observed that occasionally when no container is present the system may detect a "ghost" container and it deems the missing container requires correcting.
    b. Completely rule out blue lidded containers
       Occasionally the system will determine a blue-lidded container requires correcting. The algorithm needs to be adjusted to ignore blue containers as the corrective action causes a collision and will trip a 'protective stop' on the UR3, requiring manual intervention.
2. Use the integrated force sensor on the UR3 during corrective action.
   During corrective action, the built-in force sensor on the UR3 could be used to provide closed loop control to efficiently level down the container opposed to a generic 'stamp' action currently used.
3. Add logging to SIFMES for quality reports.
   Detailed reports from the vision inspection could be included in the SIFMES reporting system for greater analysis on the stations performance and breakdown of lead time during the packaging process.
4. Add dedicated 'vision' and 'correction' states to PLC steps code.
   The addition of dedicated states will lead to cleaner PLC code and may assist with troubleshooting if problems in service arise in the future.

Use of GenAI

This project made extensive use of generative AI to support reverse engineering, troubleshooting, and software development. GenAI tools assisted in analysing PLC Structured Text code and the UR3 script, producing summaries, flowcharts, and interpretation of state-machine interactions that significantly accelerated the reverse-engineering process. They were also used in the development of the Python scripts and portions of the modified PLC logic through carefully designed prompts. Given the complexity of the system and the limited timeframe of approximately three weeks, the project could not have been completed to this level of depth and integration without the assistance of GenAI. These tools were also used to help draft, refine, and review sections of this report. Access to selected transcripts used during development can be found here: ChatGPT transcripts

Bibliography

Universal Robots. (n.d.). UR Modbus guide (Version 0.1.4).
https://s3-eu-west-1.amazonaws.com/ur-support-site/66329/UR%20Modbus_bba_mods_0.1.4.pdf

Van den Bergh, B., Moons, T., & Van Gool, L. (2021). Automatic detection of tilted cylindrical objects using computer vision [PDF]. University of Hasselt.
https://documentserver.uhasselt.be/bitstream/1942/35035/1/2d5ba28d-db91-444c-a556-26e86038135e.pdf