Handling with AI-enhanced Robotic Technologies for flexible ManUfacturing

D1.5 – Real world scenarios for system validation

Deliverable ID:	D1.5
Project Acronym:	HARTU
Grant:	101092100
Call:	HORIZON-CL4-2022-TWIN-TRANSITION-01
Project Coordinator:	TEKNIKER
Work Package:	WP1
Deliverable Type:	DEM
Responsible Partner:	TEK
Contributors:	FMI, DFKI, AIMEN, ENG, ITRI, PCL, INFAR, ULMA, TOFAS, TCA
Edition date:	30 June 2025
Version:	3
Status:	FINAL
Classification:	PU

This project has received funding from the European Union's Horizon Europe - Research and Innovation program under the grant agreement No 101092100. This report reflects only the author's view and the Commission is not responsible for any use that may be made of the information it contains.

HARTU Consortium

HARTU "Handling with AI-enhanced Robotic Technologies for flexible manufactUring" (Contract No. 101092100) is a collaborative project within the Horizon Europe – Research and Innovation program (HORIZON-CL4-2022-TWIN-TRANSITION-01-04). The consortium members are:

1	Tekniker	FUNDACION TEKNIKER (TEK) 20600 Gipuzkoa | Spain	Contact: Iñaki Maurtua inaki.maurtua@tekniker.es
2	dfki Deutsches Forschungszentrum für Künstliche Intelligenz German Research Center for Artificial Intelligence	DEUTSCHES FORSCHUNGSZENTRUM FUEF KUENSTLICHE INTELLIGENZ GMBH (DFKI) 67663 Kaiserslautern | Germany	Contact: Vinzenz Bargsten vinzenz.bargsten@dfki.de
3	aimen TECHNOLOGY CENTRE	ASOCIACIÓN DE INVESTIGACIÓN METALÚRGICA DEL NOROESTE (AIMEN) 36418 Pontevedra| Spain	Contact: Jawad Masood jawad.masood@aimen.es
4	eng	ENGINEERING INGEGNERIA INFORMATICA S.P.A. (ENG) 00144 Rome| Italy	Contact: Riccardo Zanetti riccardo.zanetti@eng.it
5	TOFAŞ TÜRK OTOMOBİL FABRİKASI A.Ş.	TOFAS TURK OTOMOBIL FABRIKASI ANONIM SIRKETI (TOFAS) 34394 Istanbul | Turkey	Contact: Nuri Ertekin nuri.ertekin@tofas.com.tr
6	PHILIPS	PHILIPS CONSUMER LIFESTYLE BV (PCL) 5656 AG Eindhoven | Netherlands	Contact: Erik Koehorst erik.koehorst@philips.com
7	ULMA	ULMA MANUTENCION S. COOP. (ULMA) 20560 Gipuzkoa | Spain	Contact: Leire Zubia lzubia@ulmahandling.com
8	deepblue	DEEP BLUE Srl (DBL) 00193 ROME | Italy	Contact: Erica Vannucci erica.vannucci@dblue.it
9	FMI	FMI HTS DRACHTEN B.V. (FMI) NL-4622 RD Bergen Op Zoom, Netherlands	Contact: Floris goet floris.goet@fmi-improvia.com
10	TECNOALIMENTI	TECNOALIMENTI S.C.p.A (TCA) 20124 Milano | Italy	Contact: Marianna Faraldi m.faraldi@tecnoalimenti.com
11		POLITECNICO DI BARI (POLIBA) 70126 Bari | Italy	Contact: Giuseppe Carbone giuseppe.carbone@poliba.it
12	OMNIGRASP	OMNIGRASP S.r.l. (OMNI) 70124 Bari | Italy	Contact: Vito Cacucciolo vito.cacucciolo@omnigrasp.com
13	ITRI Industrial Technology Research Institute	INDUSTRIAL TECHNOLOGY RESEARCH INSTITUTE INCORPORATED (ITRI)	Contact: Curtis Kuan curtis.kuan@itri.org.tw
14	INFAR INFAR INDUSTRIAL CO., LTD.	INFAR INDUSTRIAL Co., Ltd (INFAR) 504 Chang-hua County | Taiwan	Contact: Simon Chen simon@infar.com.tw

Funded by the European Union

Document history

Date	Version	Status	Author	Description
12/05/2025	01	Draft	TEK	Definition of ToC
16/06/2025	02	Draft	DFKI, FMI, AIMEN, ITRI, ENG, ITRI, PCL, INFAR, ULMA, TOFAS, TCA	Contributions from partners
30/06/2025	03	Final	TEK	Version submitted

Funded by the European Union

Table of contents

Funded by the European Union

Funded by the European Union

List of figures

Funded by the European Union

Funded by the European Union

List of tables

No se encuentran elementos de tabla de ilustraciones.

Funded by the European Union

Acronyms

List of the acronyms	
PBD	Programming by demonstration
UCx	Use case ‘x’
RA	Reference architecture
BT	Behaviour Tree
UI	User Interface
MPC	Model Predictive Controller
GMM	Gaussian Mixture Model
URDF	Unified Robot Description Format
DMP	Dynamic Movement Primitives
FBG	Fiber Bragg Grating
SCM	Software Configuration Management

Funded by the European Union

Executive Summary

A new release of the real-world scenarios based on the analysis and requirements reported in D1.1 has been deployed.

In this release some changes in the setups have been included and all technical results have been integrated and are ready to be validated in the last six months of the project.

1 Introduction

This report is an update of the document "D1.2 Initial setup of real world scenarios", and reports the final setups and modules integrated in each demonstrator, before starting the final validation (M31-M36).

In sections 2 to 9 each final end-user demonstrator is described (the use case overview has been taken from D1.2). The last two chapters include a report on the final version of the Reference Architecture of the HARTU project and the Software Configuration Management.

2 TOFAS – UC1 – Spare parts delivery preparation

2.1 Use case overview

The spare parts delivery preparation process is divided into two sub-processes. The first is the input box preparation in the warehouse (left picture in Figure 1) and second is the order preparation in the workshop (Figure 2).

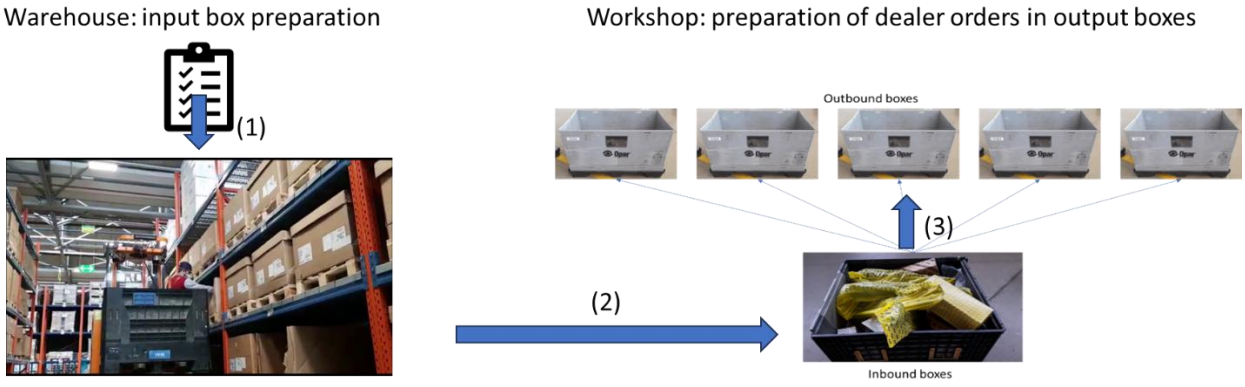


Figure 1. Input box preparation in the warehouse (1) and process overview (2)

Every day (Figure 1) (1) the warehouse operators receive a list of products that have to picked from the warehouse shelves to complete different orders for their dealers. The list consists of products of different sizes, shapes, and weights. The operators go through the warehouse on a

Funded by the European Union

fork-lift and an input box (2), take the products in the list and put them inside the box, without any particular order. Then, they go to the workshop and place the input boxes in front of the output boxes (3).

Once in the workshop (Figure 2), other operators (4) are in charge of taking the products one by one and identify them and the box with the barcode reader; then (5), they move to the corresponding output box as indicate by the barcode reader (connected to the Server through internal Wi-Fi system), identify the box with the reader and, finally, place them in the output box. They try to optimize the occupancy of the output box.

(4) (5)

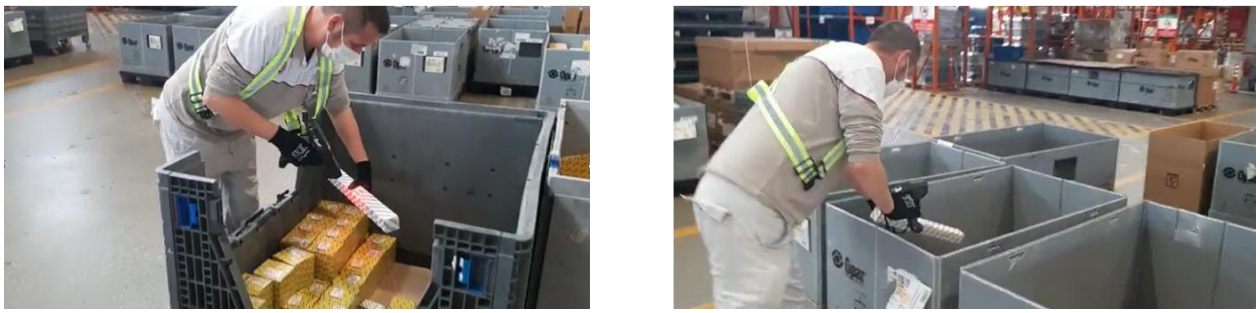


Figure 2. Order preparation at the workshop

Based on the experience gained in the Horizon 2020 PICKPLACE project this constraint is introduced:

“Inbound boxes shall only contain products that come in cardboard boxes.”

The current procedure includes the mixing of any kind of products in the same inbound box. This means that products in cardboard boxes, products in plastics bags and unpacked products are included in the same box. The presence of plastic bags of different characteristics represents a serious difficulty for grasping, either by vacuum or with 2-3 fingers, as there is no way of knowing the shape of the product inside and, depending on the plastic used, the vacuum doesn’t work.

However, 60% of products come in carboard boxes. The conclusion in PICKPLACE was that by adapting the preparation procedure in the warehouse (carboard boxes in one inbound box and those coming in plastics bags and those unpacked in another), and creating a collaborative application on the preparation shopfloor, it was possible to achieve an efficient system.

Figure 3. Example of different plastic bags used for packaging

Funded by the European Union

2.2 Prototype description and components

In this UC, and application using a mobile manipulator to handle the cardboard boxes and allow accessing multiple input and output boxes in a flexible way, has been developed.

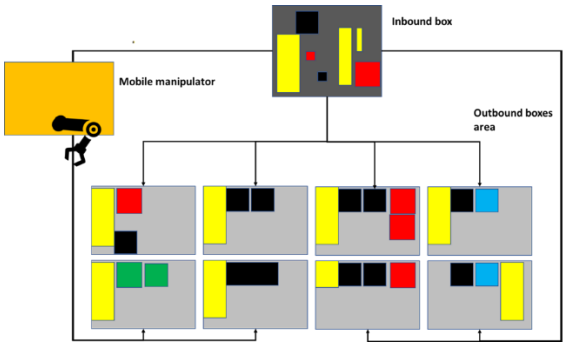


Figure 4. Layout of the proposed preparation area (TOFAS)

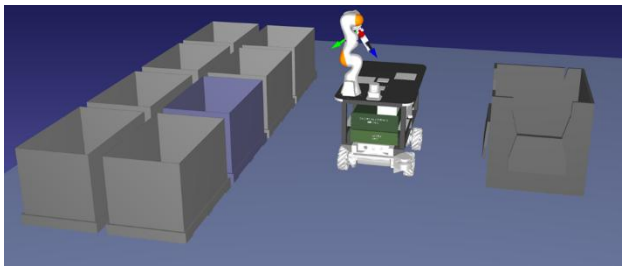


Figure 5. Simulation of TOFAS spare part order preparation scenario

A prototype has been created at TEK as a scaled-down version of the proposed overall system, due to the availability of physical space at the shopfloor.

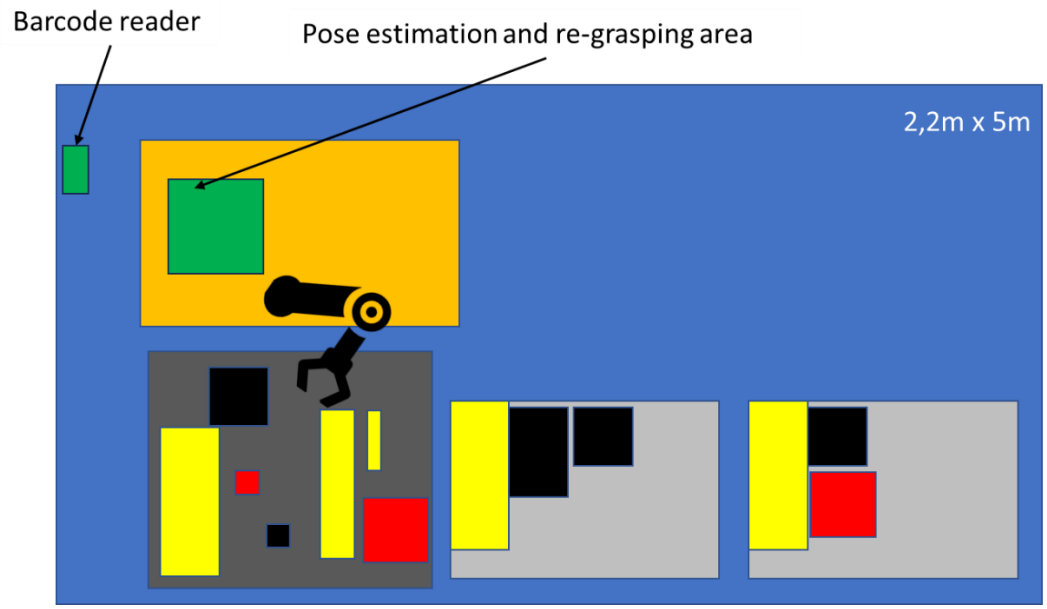


Figure 6. Scaled-down version of UC1 in TEK

Next picture shows the current UC1 prototype:

Funded by the European Union

Figure 7. UC1 prototype layout

The demonstrator consists of the following elements:

- Mobile platform: Segway RMP omnidirectional mobile + KUKA IIWA 14.
- A ZIVID camera mounted on the end of arm instead of the initial ZED camera. The corresponding image acquisition node has been developed

Figure 8. ZIVID Camera

- 3 suction grippers for handling different size boxes

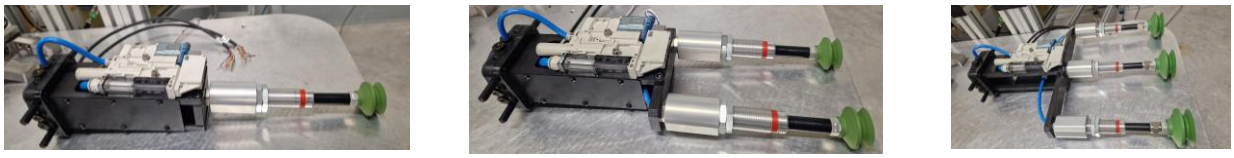


Figure 9. Suction tools for TOFAS-UC1

Funded by the European Union

- Tool change station embarked on the robot for automatic tool change.

Figure 10. Tool exchange station for TOFAS UC1 and UC2a

- Identification System.

  All cardboard boxes have a barcode that has to be read to know the reference of the product inside the box and to be able to know which order it corresponds to. In addition, as it is requested to create a mosaic in the output box (it is not enough to drop the products), it is necessary to control the way the product has been grasped. To achieve these objectives:

  - The robot holds the carboard box 600mm above a barcode reader (model ZEBRA FS40-WA50F4-2100W) facing upwards.
  - The robot (1) leaves the carboard box on the platform surface, (2) take a picture with the built-in camera, (3) estimates the pose and picks it up again.

Funded by the European Union

Figure 11. Barcode reader in the demonstrator

The sequence of actions in the demonstrator are:

1. The robot generates an image of the input box (this is done only on the first iteration; for the rest, the image is taken in step 5).
2. Identifies the best candidate (based on the grasping points and the gripper on the arm).
3. If necessary, the robot changes the gripper.
4. Picks the object and places it in front of the barcode reader.
5. The robot places the box on the platform surface and using the built-in camera, estimates the pose, and picks it up again. Before picking, it takes an image of the inbound box.
6. The control system provides the destination box according to the order to which it corresponds, and the label read.
7. The robot picks the part and navigates to the destination output box.
8. The robot calculates the position inside the output box and executes the release.
9. The robot takes an image of the output box once the part has been released.
10. The robot navigates to the position of the input box.
11. A new iteration starts

Funded by the European Union

2.3 System control and GUI

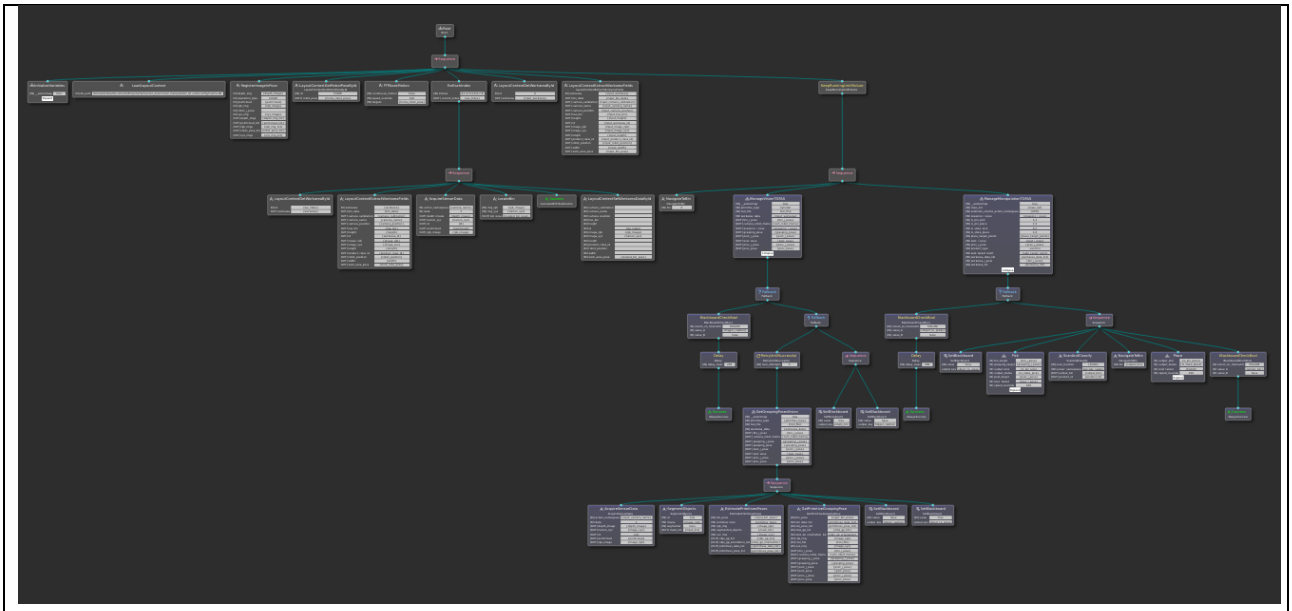


Figure 12. BT for UC1 demonstrator control

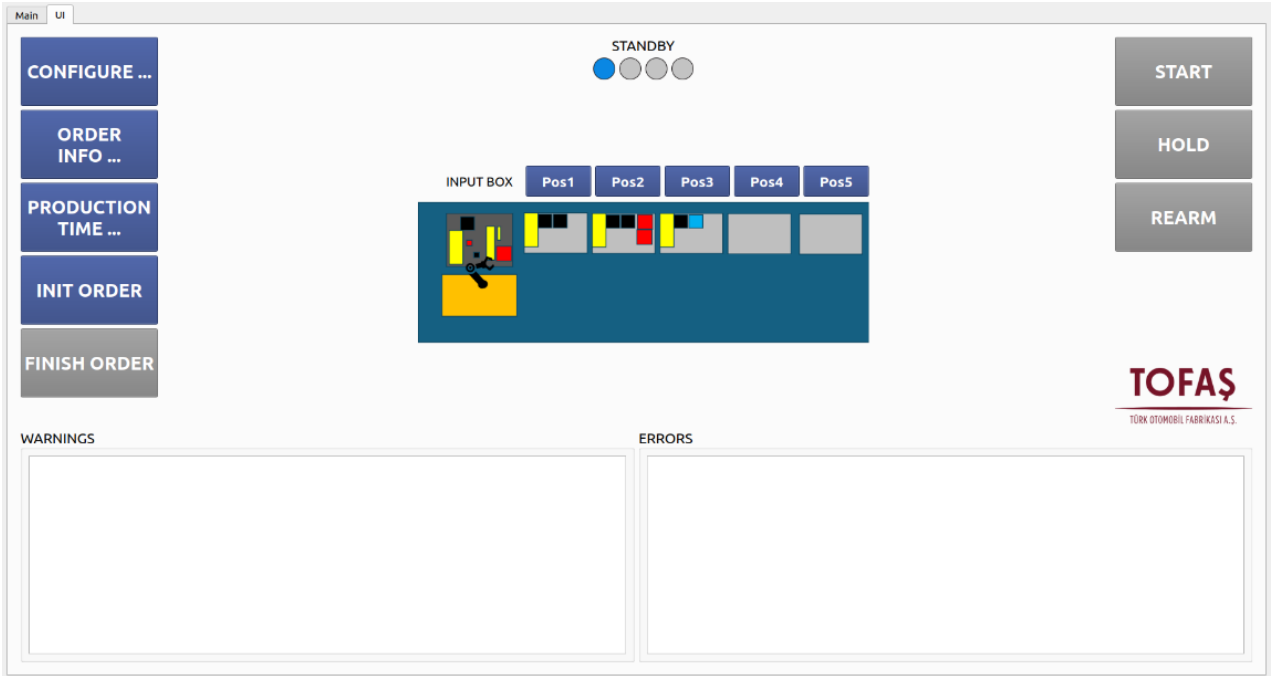


Figure 13. GUI for UC1

Funded by the European Union

3 TOFAS – UC2a – Kitting in the automotive sector

3.1 Use case overview

This use case corresponds to the preparation of kits of components, an operation known as 'kitting' in the automotive sector. The subsequent pre-assembly step is treated as a separate use case: UC2b pre-assembly at the corresponding assembly workstation.

In the kitting area, products are taken from containers/boxes in which they can be arranged in two main configurations: (1) Product-specific individual compartments (Figure 14); (2) Semi-structured configuration (Figure 15), forming layers that are separated by means of separators (cardboard or plastic); randomly distributed products (Figure 16) are not included in the kitting operation, but are managed by the operators at the assembly station.

Figure 14. Products in special compartments

Figure 15. Products in semi-structured configuration

Figure 16. Products randomly distributed

Components that must be included in the kit are placed in containers on one side of the preparation area, and the destination containers on the other side, as shown in the next pictures. The only exception is the discs, which are placed inside blue plastic boxes on a shelf near the conveyor belt.

Funded by the European Union

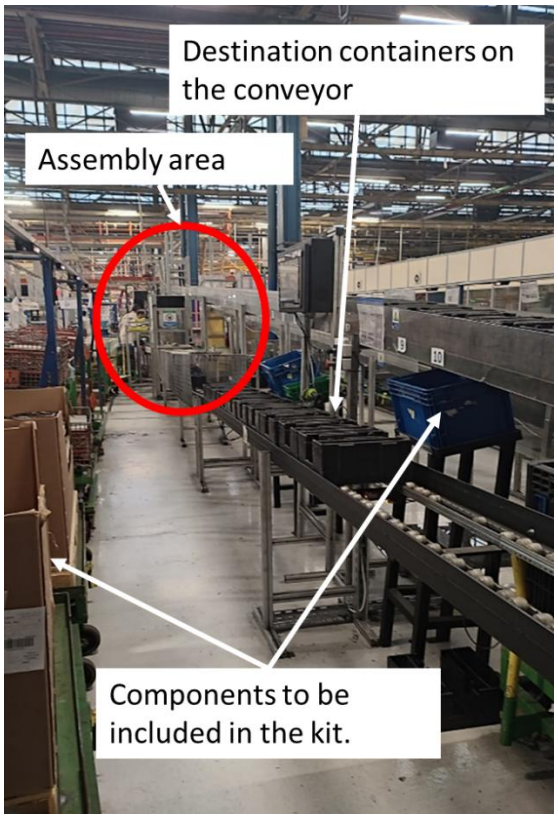


Figure 17. Kitting preparation area.

Figure 18. Destination containers on the conveyor. The blue boxes contain the discs

To start the preparation of kits, the operator presses the button in Figure 19 and 10 empty output containers arrive at the preparation area on the conveyor belt.

Figure 19. Push buttons to control the conveyor belt that transports the output containers

Figure 20. Pick to light device above each input container

Then, the operator starts the pick-and-place process: on top of each input container there is a pick-to-light device that shows the destination conveyor for each component (Figure 20). The operator takes the component, leaves it in the corresponding output box on the conveyor and acknowledges the action on the pick-to-light device.

Funded by the European Union

Finally, 10 paper forms are taken and introduced in each destination box, and the operator presses the button to transport the full containers on the conveyor belt to the assembly workstation (at the back of Figure 17).

Figure 21. Operator taking a component from the input container (left side). It is then placed in the target container (right side)

In this UC, it is proposed to use the same mobile manipulator as in UC1 to handle the products, as it provides a more flexible alternative to other solutions like a robot mounted on linear tracks.

A prototype has been created at TEK and, after integration and validation of the HARTU results, it can be transported to the TOFAS plant in Bursa for final demonstration.

3.2 Prototype description and components

The demonstrator installed at TEK is a scaled-down version of the proposed overall system, due to the availability of physical space at the shopfloor.

Next picture shows the current UC2 prototype:

Funded by the European Union

Figure 22. Scaled-down version of UC2a in TEK

Figure 23. Detail of the output containers on the table

The demonstrator consists of the following elements:

- Mobile platform: Segway RMP omnidirectional mobile + KUKA IIWA 14.
- A ZIVID camera mounted on the end of arm instead of the initial ZED camera.
- 4 Grippers
    - Two magnetic.
    - One 2-finger.

Funded by the European Union

- One 3-finger.

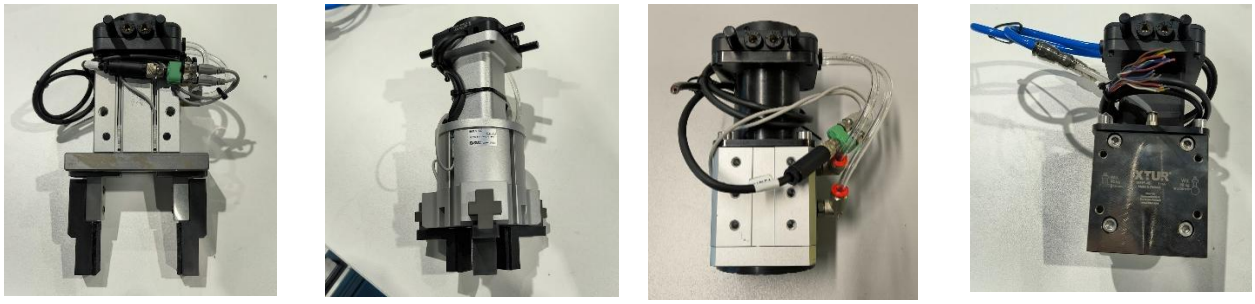


Figure 24. Tools for the TOFAS UC2a

- Tool change station embarked on the robot.
- Workpiece repositioning station

  Part repositioning station. Parts are picked using a magnetic gripper, but the placement in the output container requires to re-pick the part in a different way using a 2/3 finger griper.

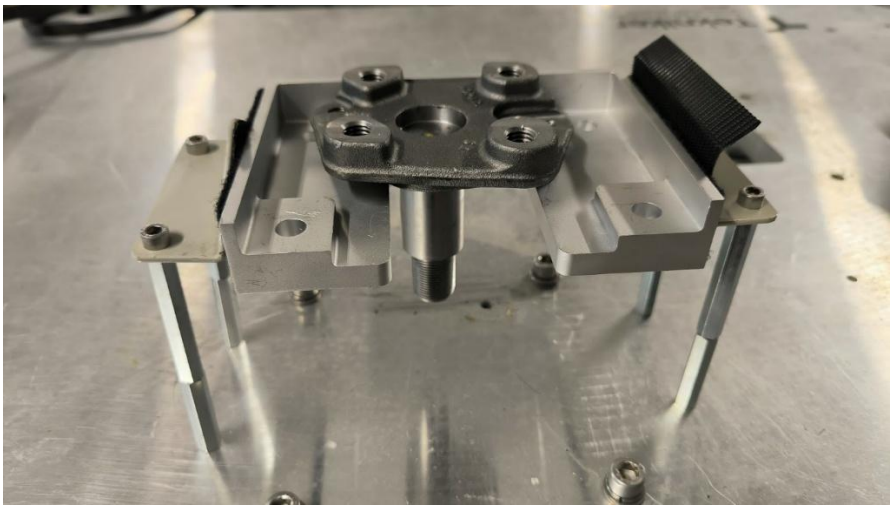


Figure 25. Part re-grasping station

The sequence of actions in the demonstrator are:

1. The robot receives the list of items to be placed in each output box. For each of them:

Funded by the European Union

2. It navigates to the input container.
3. It takes an image and identifies the best candidate.
4. It picks the object with the magnetic gripper and places it on the repositioning station.
5. The robot navigates to the destination container.
6. It takes an image of the area and locates the container accurately.

   The robot picks the part from the repositioning station and places it on the corresponding output container.

Steps 2 to 6 are repeated until the order is completed

When necessary, the robot changes the gripper.

Figure 26. First integration tests for UC2a, tool exchange

3.3 System control and GUI

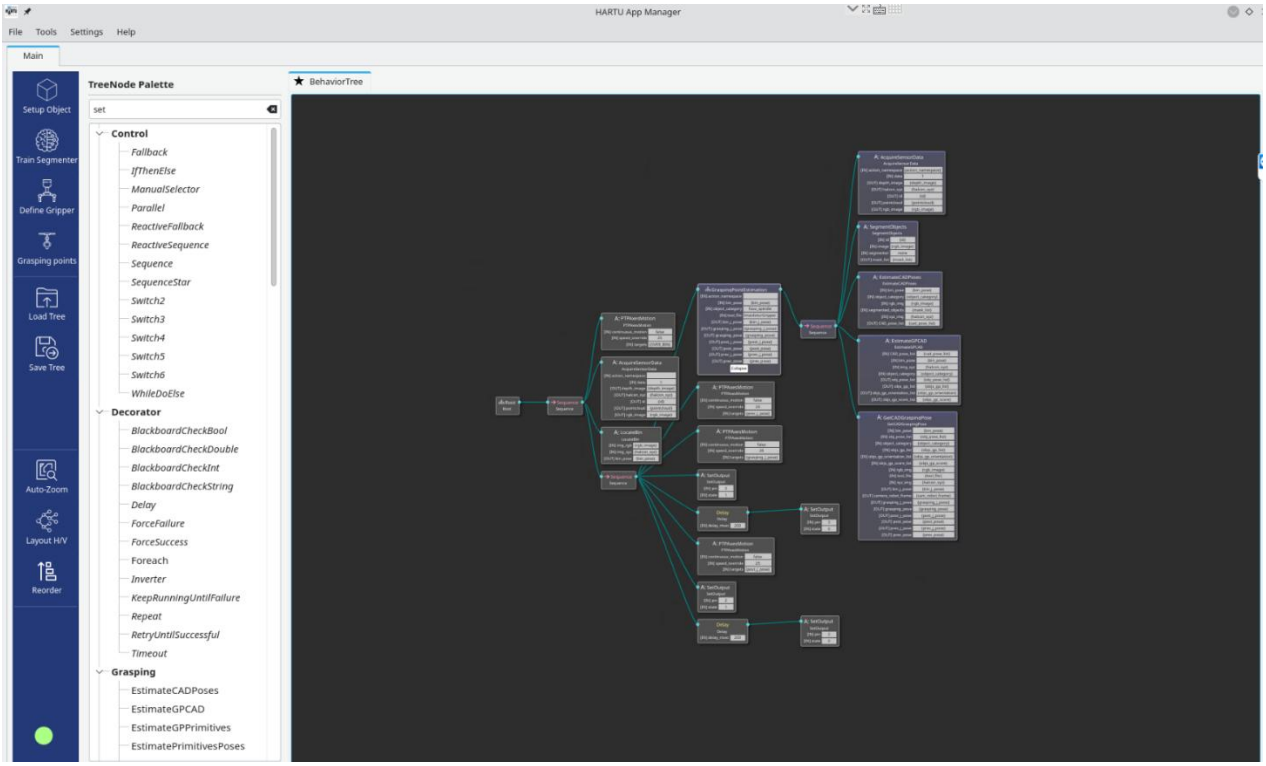


Figure 27. BT for UC2-a demonstrator control

Funded by the European Union

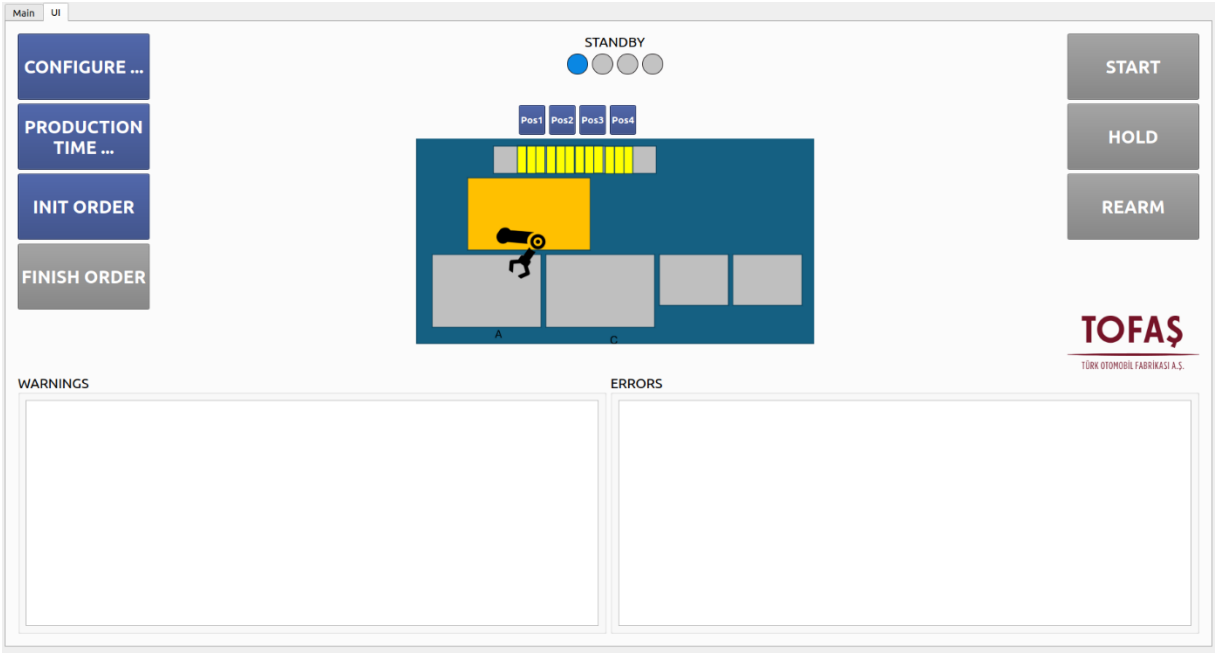


Figure 28. GUI for UC2-a

4 TOFAS – UC2b – Pre-assembly in the automotive sector

4.1 Use case overview

This use case corresponds to the pre-assembly of components after the kitting operation described in UC2a. The use case to be considered is the pre-assembly of rear wheel units such as shown in Figure 29. It shows the pre-assembly of the rear-wheel drum brake after the hub has been installed on the spindle, which includes the following steps (from left to right): (1) Washer loading, (2) Nut loading, (3) Nut pre-screwing, (4) Drum loading, (5) Pre-screwing, (6) Screwing.

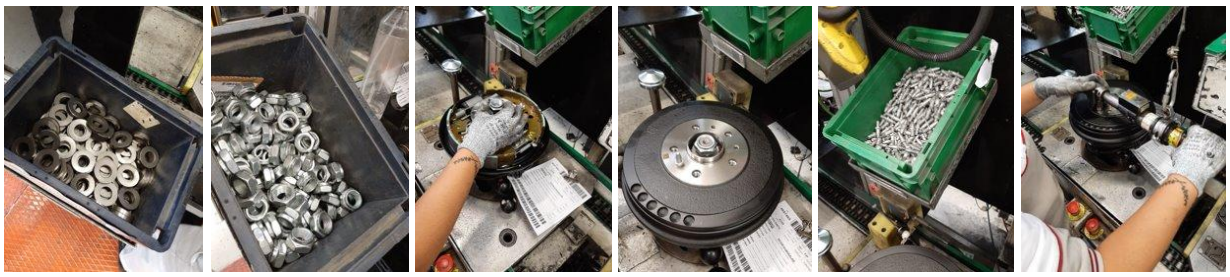


Figure 29 Use case UC2b, pre-assembly of real wheel in automotive sector

Funded by the European Union

4.2 Prototype description and components at DFKI

The laboratory prototype set up at DFKI Robotics Innovation Center comprises two rigidly mounted KUKA iiwa industrial manipulators that can be equipped with the available Robotiq 3-F and Robotiq 2-F grippers and an Onrobot 3FG25 purchased for this use case.

The Onrobot 3FG25 is aimed at handling the heavy cylindrical parts of this use case. For this purpose, custom finger extensions have been manufactured inhouse. The setup is shown in Figure 30. A ROS2 node has been created to interface the gripper and to carry out the conversion from target diameter with custom fingers to finger angle and gripper command. Both manipulator arms are additionally equipped with an end-effector force-torque sensor.

Optionally, RGB-D Cameras are available to provide images and point clouds for object detection as well as two Sick laser scanners for workspace monitoring.

A RGB-D camera in the top center of the system has been integrated to detect gestures by the user that control the gripper opening with the distance between the user's thumb and index finger. The manipulator arms, grippers and sensors have been connected to a control PC, where software drivers have been set up to interface with the hardware in ROS2.

The table surface area of the setup can be used as assembly area, or an additional table can be used next to the existing one.

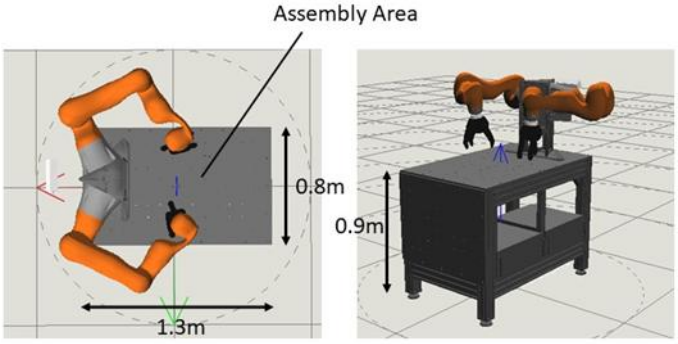


Figure 30. Setup

Funded by the European Union

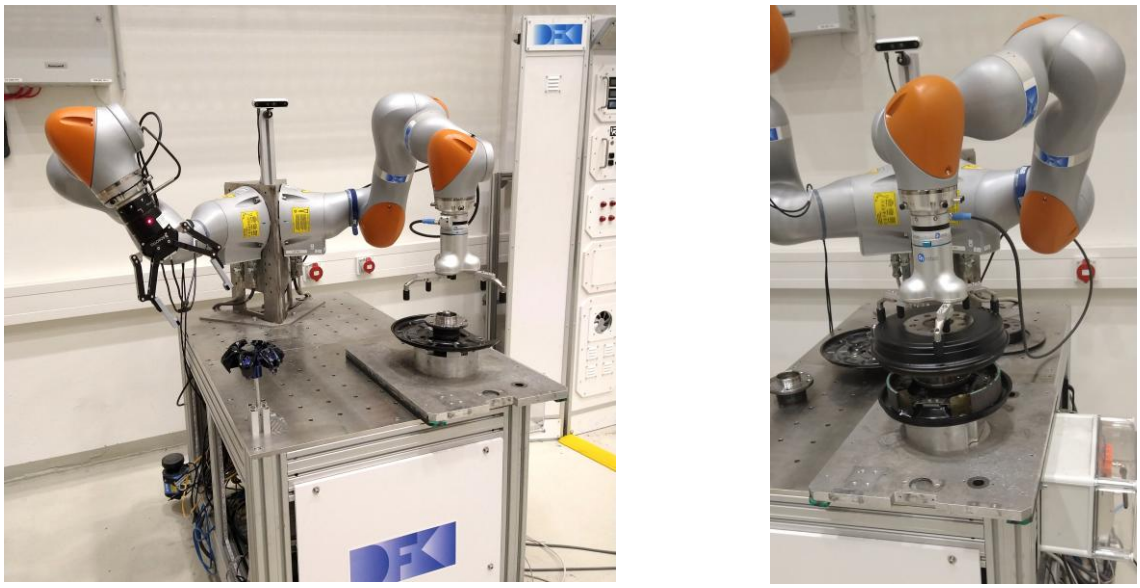


Figure 31. Dual-Arm KUKA iiwa as demonstrator for UC2b. Upper image: Dimensions; left image: actual installation; right image: gripper holding the drum brake (approx. 5.5 kg) in place for assembly on the brake unit (brake shoes, hub, spindle, back plate).

4.3 System control and GUI

For the execution of the learnt motion and end-effector force reproduction, a behavior tree node for integration into the HARTU AppManager has been created. However, for recording the user demonstrations a dedicated GUI shown in Figure 32 has been created.

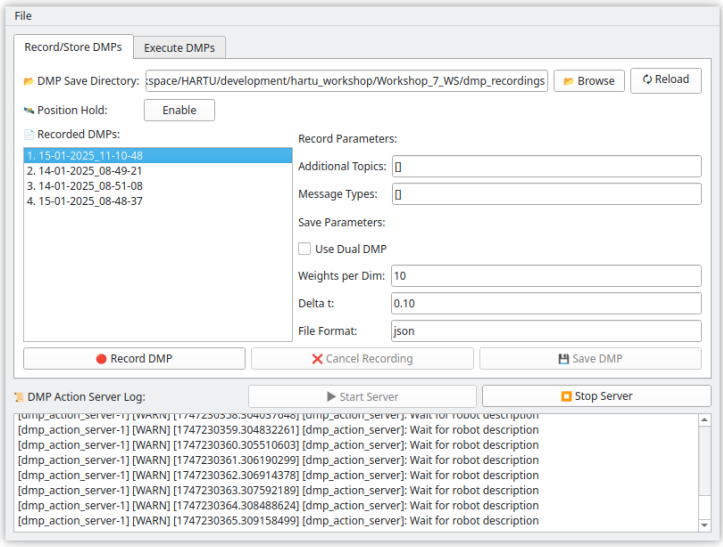


Figure 32 GUI for recording user demonstrations

Funded by the European Union

5 PCL – UC3 – Handling for mass customization in the consumer goods sector

5.1 Use case overview

Philips is a world leader in mass production of consumer goods. To do this efficiently, the current processes are designed for high volumes with little product variation. Currently there is a trend towards more personalization. This results in more product variation within a given process, requiring the equipment to be more flexible and also easily reconfigurable.

A typical example of customization is the lacquering line, where parts are coated with a layer of lacquer to match the product design to the consumer's need. A wide range of products is fixtured manually by operators on the jigs that are going into the spray booths.

Image – fixturing at a loading/unloading station

Image – loading/unloading stations

The scope of the HARTU prototype is the automation of the insertion and removal of parts on/from the lacquering jigs.

This prototype will demonstrate flexibility, as it should be able to work with different product variants and colors (e.g., chest pieces and front panels). Furthermore, it should be easy to reconfigure or train the system for new product introductions.

Image – Chest panel

Image – Front panel

Funded by the European Union

In addition, this prototype shows the adjustability of a complex fixturing motion to a changing environment (i.e., a freely rotating jig). The placement of the parts on the jig is a complex wrist motion, that is different for each position on the jig. The parts are attached to the jig using a snap-fit connection. Correct placement can be checked by an audible click of the snap-fit joints.

Finally, the removal of parts from the jigs must always be done carefully to avoid scratching the newly applied surface finish.

5.2 Prototype description and components at PCL

The demonstrator to be installed at PCL focuses on the placement and removal of products on the lacquering line. To demonstrate the repeatability of the action, this is done in a continuous loop. Products are picked from a single tray and placed back in the same tray with the support of the perception system.

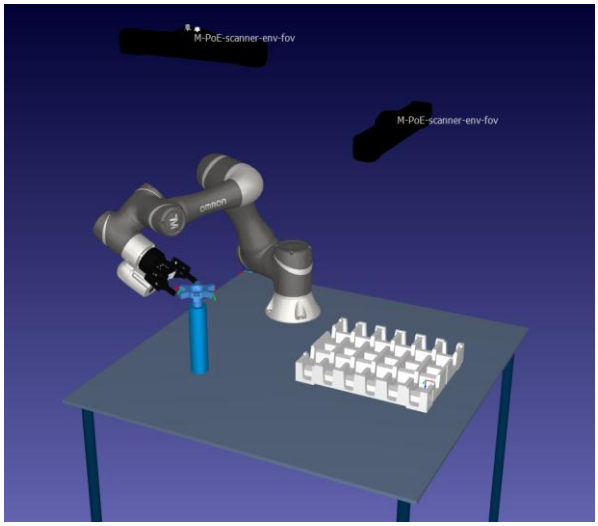


Figure 33. Side view proposed set-up

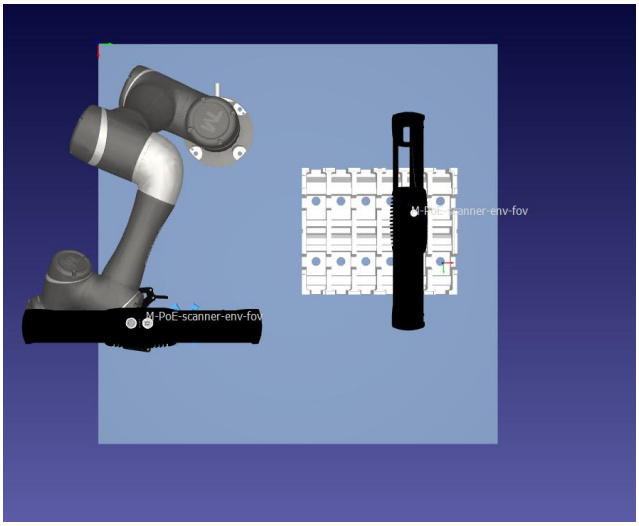


Figure 34. Top view

The state of the current set-up is as shown in Figure 35.

Funded by the European Union

Figure 35. Current setup

The setup at PCL consists of the following parts:

- 6-axis robot with 700mm reach. Type: Kuka kr6 R700 sixx
- 2 Structured-light 3D Scanners. Type: Photoneo PhoXi 3D scanner M
- Force torque sensor. Type: Robotiq FT300-S
- 2-Finger servo-electric gripper. Type: Weiss CRG 30-050
- Gripper fingers. Type: Custom 3D printed fingers

The sequence of steps for this prototype is as follows:

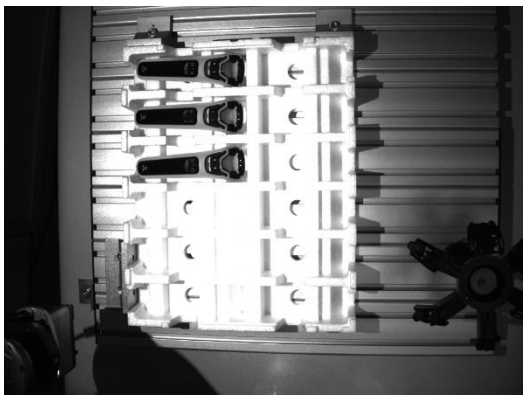
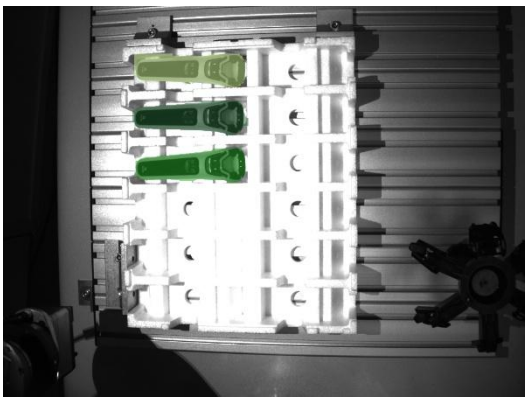
A. Place the pieces until the jig is full:
   1. Grab image of tray.
   2. Estimate pick pose.
   3. Pick product from tray.
   4. Grab image of jig.
   5. Estimate place pose.
   6. Place product on jig.

B. Remove the pieces until the jig is empty.
   1. Grab image of jig.
   2. Estimate pick pose.
   3. Remove product from jig.
   4. Grab image of tray.
   5. Estimate place pose.
   6. Place product in tray.

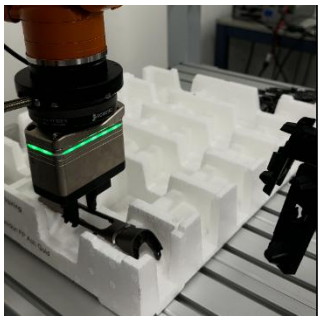
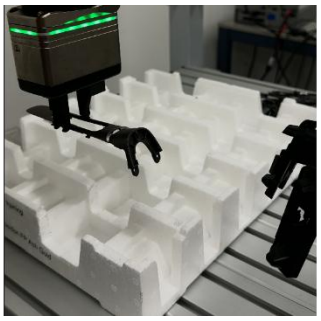
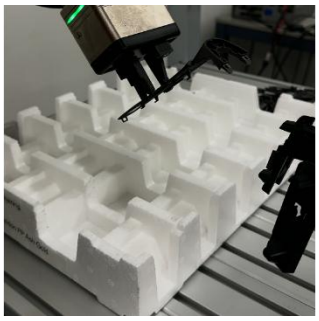
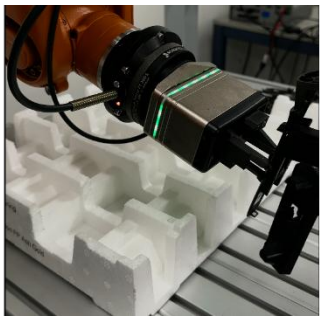
Funded by the European Union

Data acquisition and testing:

- Image acquisition:

- Recording contact forces and trajectories for learning by demonstration:

Upcoming and potential set-up changes:

- The system is controlled by a program running in a PLC and the App Manager
- The robot is a Kuka Kr6 R700 six with an electrical gripper
- In the prototype the jig is in a fixed position

5.3 Prototype description and components at DFKI

The laboratory prototype setup at DFKI Robotics Innovation Center consists of a rotating jig, mounted on a table and a transport box with components. For robotic assembly, the initial setup consisted of a mobile platform incorporating a Franka Emika Panda robot with a two-fingered gripper shown in Figure 36. However, after a mechanical failure it could not be repaired, because the manufacturer Franka does not supply the required spare parts and assistance. Therefore, the KUKA iiwa system shown in Figure 39, which is also used in use case UC2b, has been modified to handle use case UC3. For this purpose, 2-finger grippers and force-torque sensors have been integrated along with the software components needed to operate them from the ROS2 architecture.

Similarly to the Franka Emika manipulator arm, the solution using the KUKA iiwa manipulator arms provides the possibility to investigate more complex and dynamic interaction during the assembly task due to the use of torque-controlled robot joints.

Funded by the European Union

A RGB-D camera in the top center of the system has been integrated to detect gestures by the user that control the gripper opening with the distance between the user's thumb and index finger.

In addition, new gripper fingers have been designed and manufactured to improve the handling of the delicate parts of use case UC3 with the DFKI laboratory setup. Except one initial iteration, 3D-printed design approaches have been followed for these fingers. The designs also include the integration of sockets and bolts into the 3D-printed parts. Snapshots of the design iterations of the custom fingers and using them to grasp parts of this use case are shown in Figure 38.

Figure 36. Initial PCL UC3 prototype using a Franka robot, prototype that had to be replaced due to mechnical failure (DFKI)

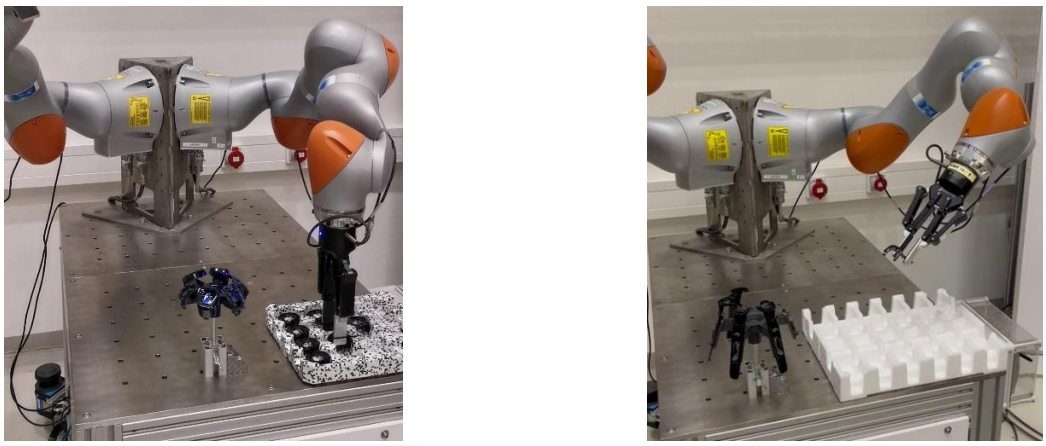


Figure 37. Current prototype setup using a KUKA iiwa manipulator arm for UC3 at DFKI, shown for two of the use case's parts

Funded by the European Union

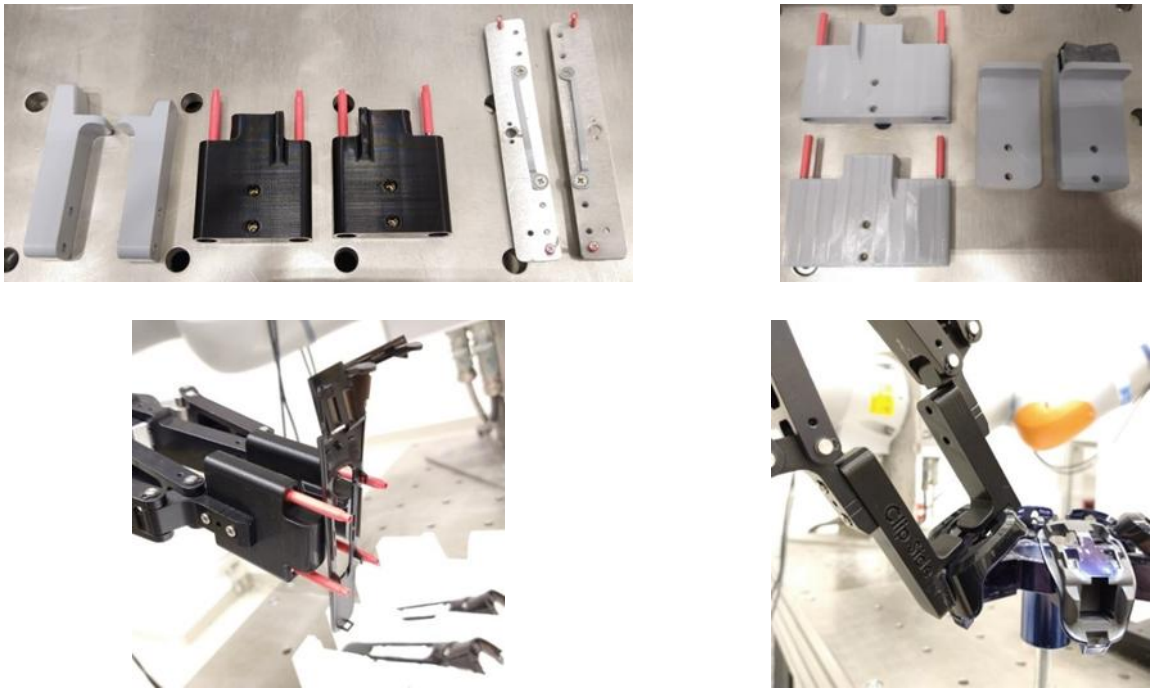


Figure 38. Multiple iterations of gripper fingers for the PCL use case (top) and depiction of how the parts are grasped with these designs (bottom)

We use kinaesthetic teaching (hand guiding) to acquire end effector trajectories and allow the operator to teach new robot skills without explicit programming. The prototype is used to investigate different approaches for representing, learning, and controlling contact-rich assembly skills, as well as to develop generic robot software. Examples of such user demonstrations are shown in Figure 39 (current setup) and Figure 40 (initial prototype setup). The developed learning-from-demonstration approach incorporates an extension which also allows the recording and reproduction of end-effector forces in addition to the motion.

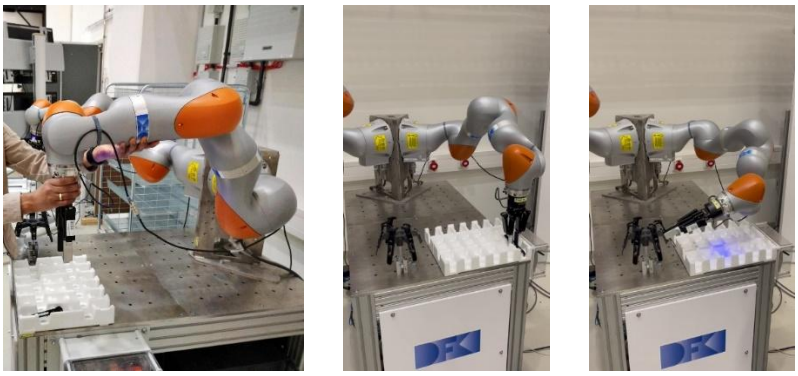


Figure 39. Programming-by-Demonstration and execution of the recorded DMPs with the current prototype setup using KUKA iiwa manipulator arms for assembly of in the PCL use case

Funded by the European Union

Figure 40 Kinaesthetic Teaching of an assemby task using the initial prototype that has been replaced due to mechnical failure

5.4 System control and GUI

The HARTU App manager has been installed on the PCL demonstrator. Integration tests have successfully been performed to validate the control interfaces for the Photoneo cameras, Kuka robot driver and Weiss gripper.

We're currently testing behaviour trees to optimize multi-part workflows, moving on from the initial single-part process flows. The goal is to identify potential cycle time improvements for testing in a relevant operational environment.

In addition, the control structure is being evaluated to ensure compliance with the Machinery Regulation (EU) 2023/1230.

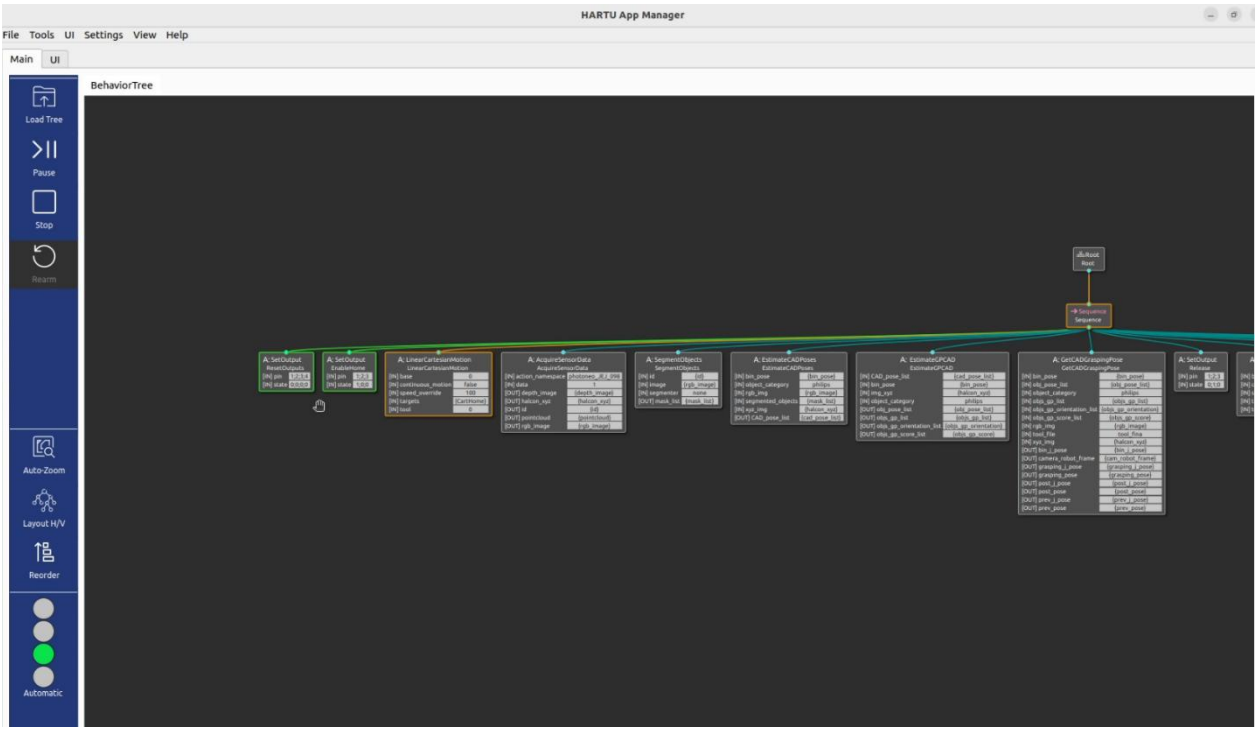


Figure 41. BT for UC3 demonstrator control

Funded by the European Union

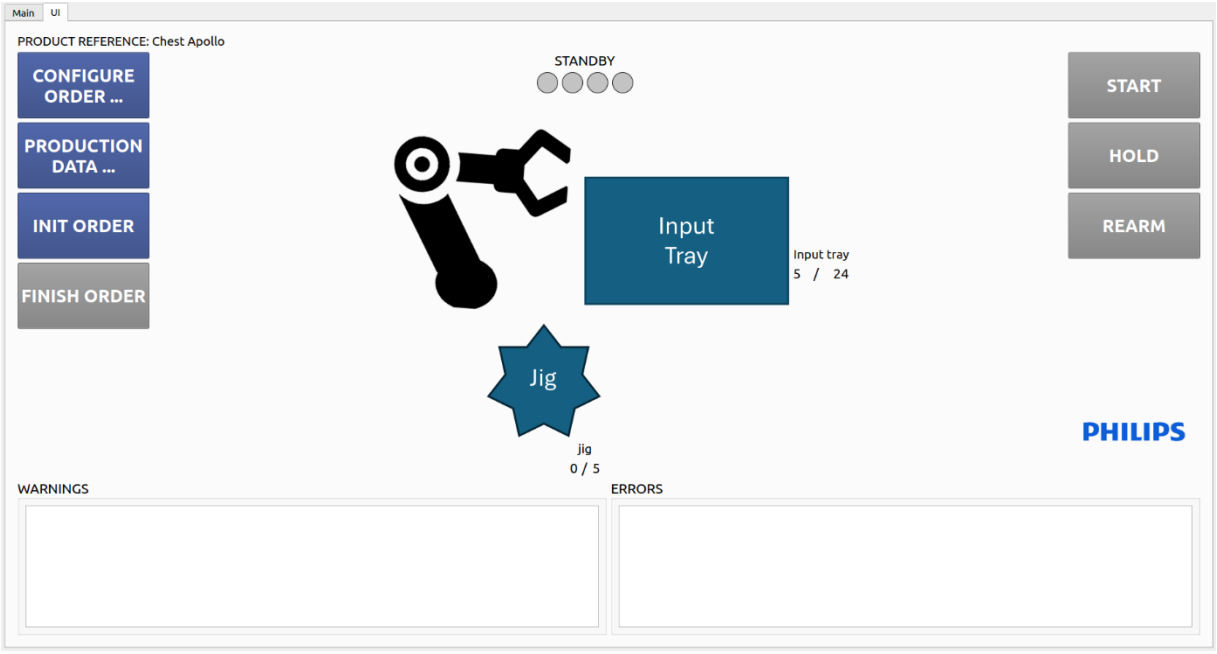


Figure 42. GUI for UC3

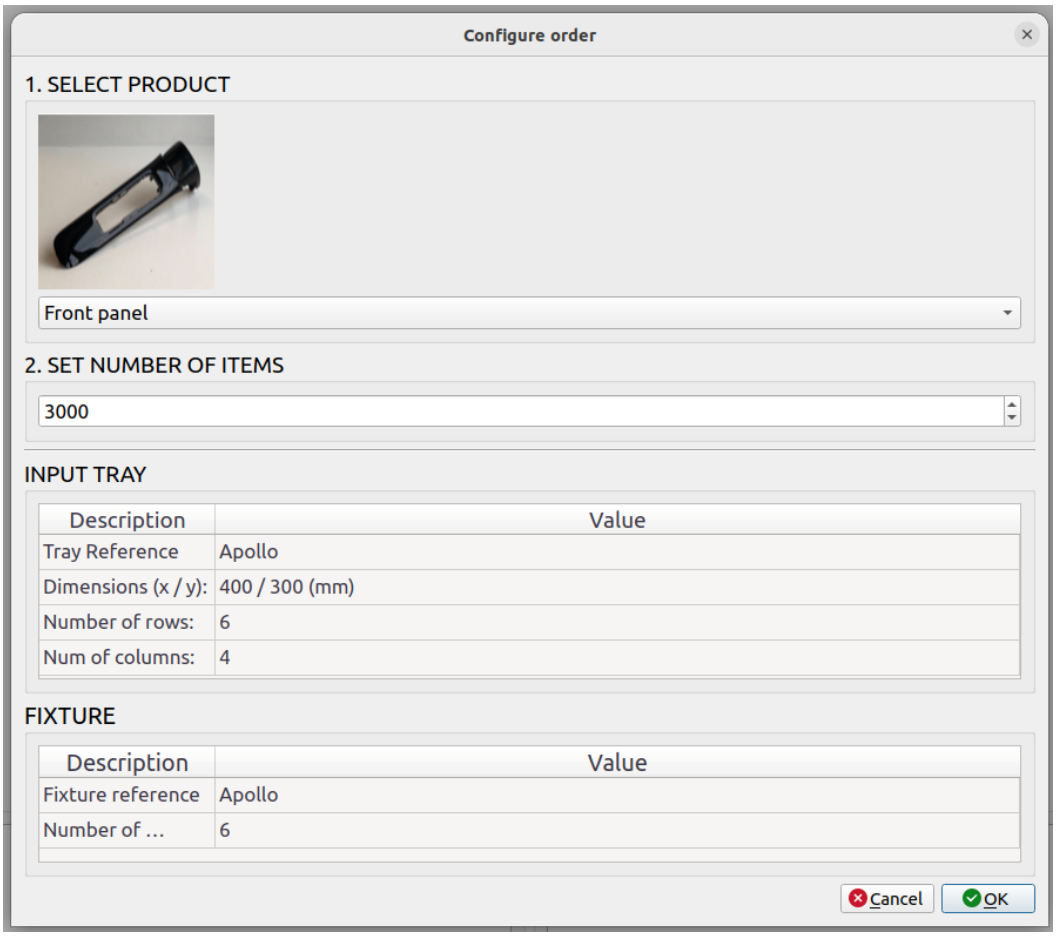


Figure 43. Order configuration in UC3

Funded by the European Union

6 TCA – UC4 – Packaging operation in food sector

6.1 Use case overview

This use case corresponds to the sorting and packaging of horticultural products. The products arrive in bulk boxes and bins of different sizes directly to the processing area from the field where they have been harvested. Then, in the production line, operators placed the products, in orderly order, into a box according to their size and shape (sorting based on other quality factors is out of scope of this project).

Figure 44. Current process for zucchini sorting

This use case implements the robotized operation for three products: Zucchini, Eggplants and Tomatoes.

To define this use case it has been taken as a reference Centrolazio, a cooperative that produces a wide variety of vegetable products. It integrates the production of the associated farms, which cover a total area of more than 300 hectares cultivated between Anzio, Latina and Sabaudia.

The procedure to be followed is as follows:

- The line operator places the input and output boxes in the corresponding station of the demonstrator.
- An image is taken from the top of the input box. The most suitable product to be picked is identified using the grasping point identification component.
- The robot generates the trajectory to pick the selected product and, once picked, moves it to the size/shape inspection area, where a new image is acquired.
- The system classifies the product according to size and shape and asks the robot to place it in the corresponding output box.
- Once placed in the box, an image of the interior of the output box is acquired for the target position of the next product that goes to this box.

Two prototypes are created at TEK and AIMEN facilities. One of them may subsequently be transferred to Centro Lazio.

Funded by the European Union

6.2 Prototype description and components at TEK

The demonstrator physical implementation is shown in next figures:

Figure 45. TCA Prototype concept implemented at TEKNIKER

The demonstrator consists of the following elements:

- FANUC CR-14iA / L on a table. There is no specific requirement to apply for a collaborative robot, but this has been selected due to availability.
- 5 ZED2i cameras, one on top of each input-output box. No camera above the rejection box. There is one NVIDIA Jetson Nano connected to each camera through USB. These mini-pc's are connected to the central PC through ethernet.
- One central PC to control the system.
- A safety radar to stop the robot in case a human being enters the robot's working area of the robot.
- A vacuum gripper, with different suction cups depending on the product to be handled. The new gripper concept developed by OMNI and POLIBA can be integrated.

Figure 46. PSEN rd1.2 safety radar sensor

Funded by the European Union

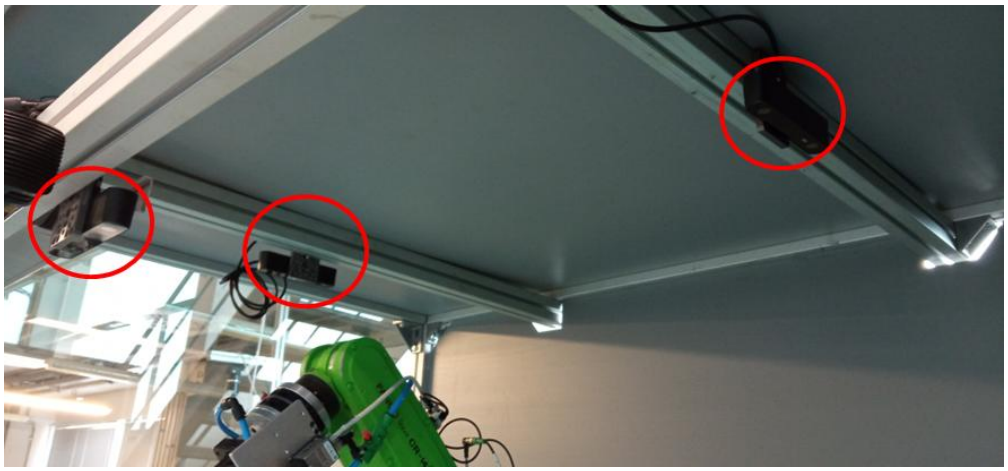


Figure 47. Detail of 3 of the ZED2i

Figure 48. Vacuum gripper

The sequence of actions in the demonstrator are:

1. The operator places an input box with bulk fruit.
2. An image of the input box is captured.
3. The system decides which fruit to pick.
4. The robot picks the fruit and places it in the position that allows an image to be taken from the side and from above.
5. The system measures the fruit and sends the destination box to the robot.
6. The robot drops the product into the destination box.
    a) Creating a mosaic (OK products)
    b) Dropping it (NOT OK products)
7. The process is repeated until the input box is empty. A message informs the operator to change the input box. Alternatively, a light can be switched on.
8. If somebody enters in the safety zone, the robot stops.

6.3 Prototype description and components at AIMEN

The main objectives of the demonstrator are:

1. Provide the realistic/near representation of the TCA use-case in terms of the working heights, lighting conditions, container sizes and vegetables.
2. Provide the necessary infrastructure to mount the developing cameras, robot (UR10e), grippers, lighting, etc. for the flowless development of the perception module (T2.2) and FBG based continuous monitoring (T4.1) technologies for all HARTU use-cases.
3. Provide platform to integrate the HARTU results on perception, grasping, and continuous monitoring related with all HARTU use-cases in general and TCA use-case in particular.

Funded by the European Union

4. Define accurate technology requirements for the TCA use-case. In terms of deployment space, human operator needs, equipment parameters, etc.
5. Provide ability to verify and validate the technology with the help of the pre-deployment performance indicators.

The demonstrator at AIMEN is shown in Figure 49.

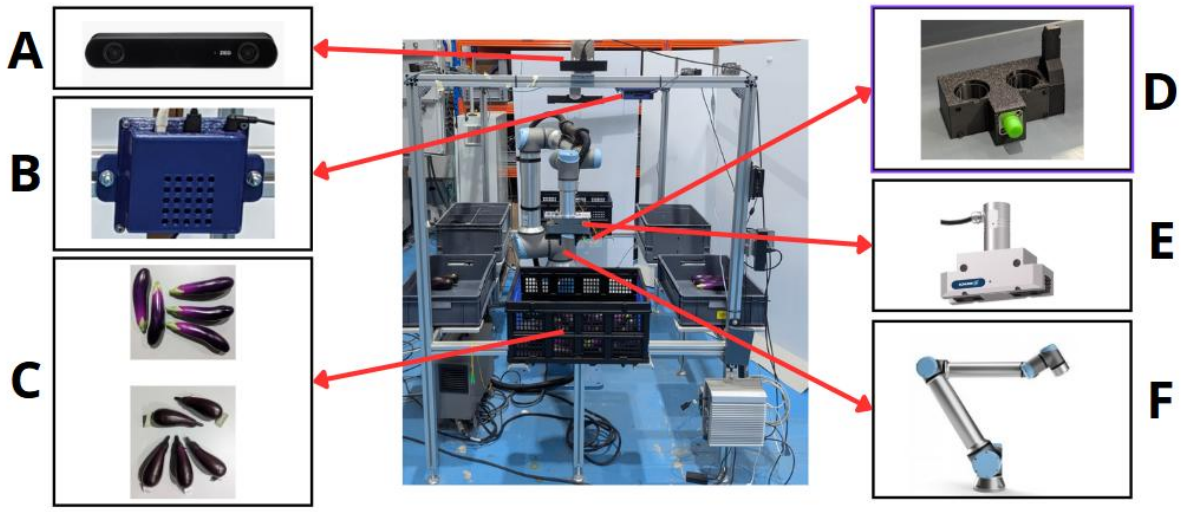


Figure 49. TCA demonstrator at AIMEN. A: Zed2i camera; B: Edge controller; C: Dummy Eggplants in container; D: Finger with embedded FBG E: Schunk EGN gripper; F: Ur10 robot

This demonstrator consists of the following elements:

- Collaborative Robot UR10e.
- Nvidia Jetson Nano with cover.
- Inbound and outbound containers.
- Zed 2i camera above inbound box (perception database of inbound eggplants).
- Aluminium profile-based metal structure.
- Led lamps
- SBI Comms interrogator
- Schunk EGN gripper equipped with FBG fingers.

Sequence of operations are similar to that described for the TEK demonstrator.

This set-up at AIMEN is being used for the following activities:

- Demonstrator for integrating HARTU results (T1.5)
    - Computer vision setup -> acquiring images: Perception module.
        - The six Zed2i cameras work as the edge perception module and send the processed information to the central PC.
        - The Zed2i cameras can be moved along the infrastructure.

Funded by the European Union

- The height of the infrastructure can also be adjusted.

- Robotics setup -> acquiring forces/trajectories/etc: Grasping module and Continuous monitoring module.
  - Universal robot is installed at the centre of the demonstrator.

6.4 System control and GUI

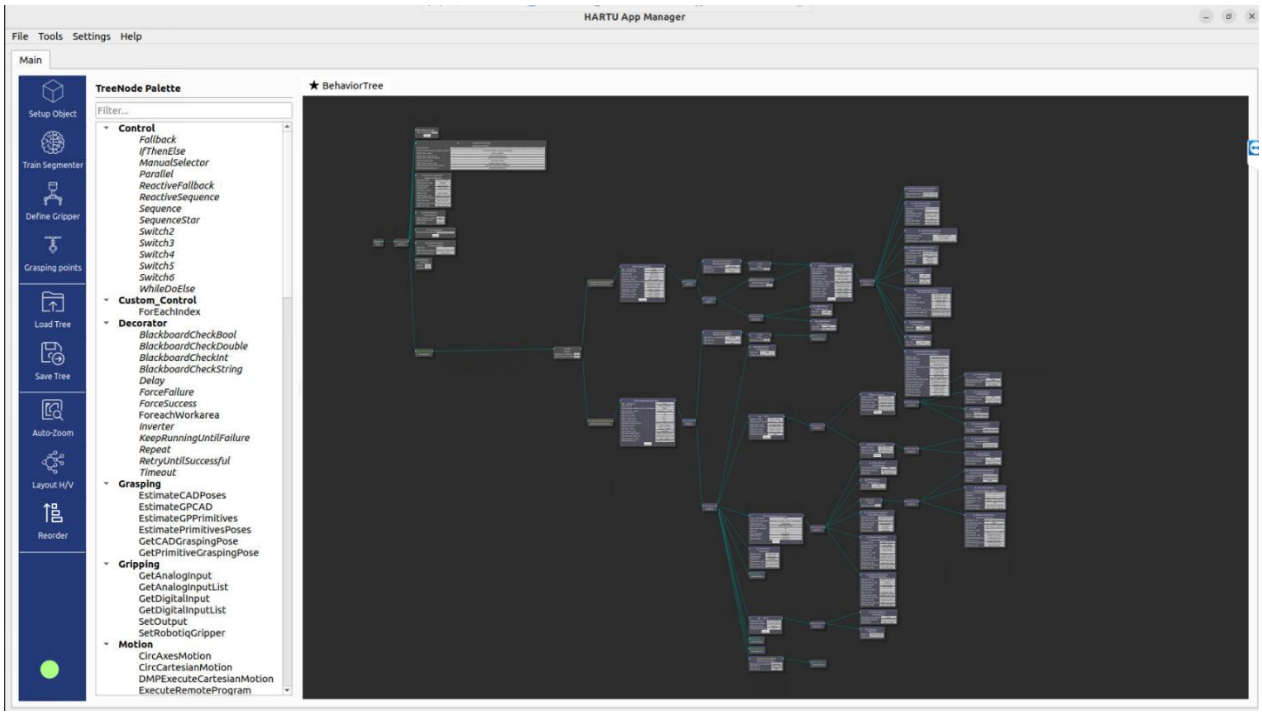


Figure 50. BT for UC4 demonstrator control

Funded by the European Union

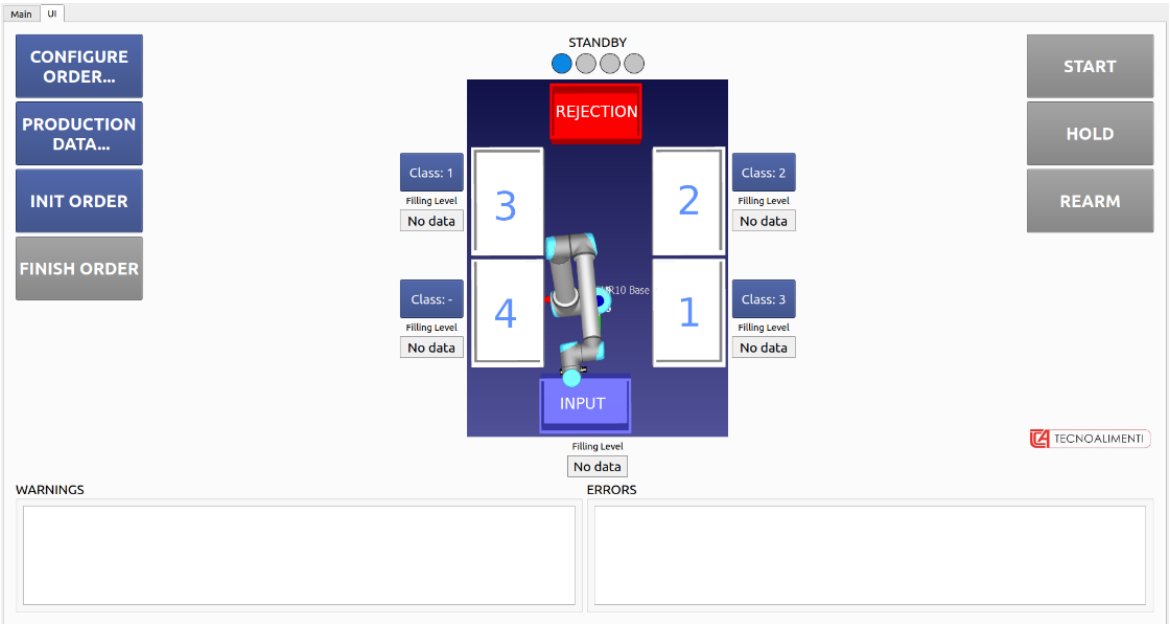


Figure 51. GUI for UC4

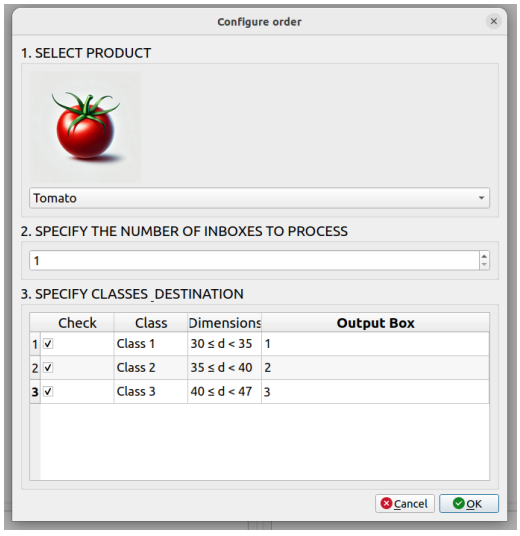


Figure 52. Order configuration in UC4

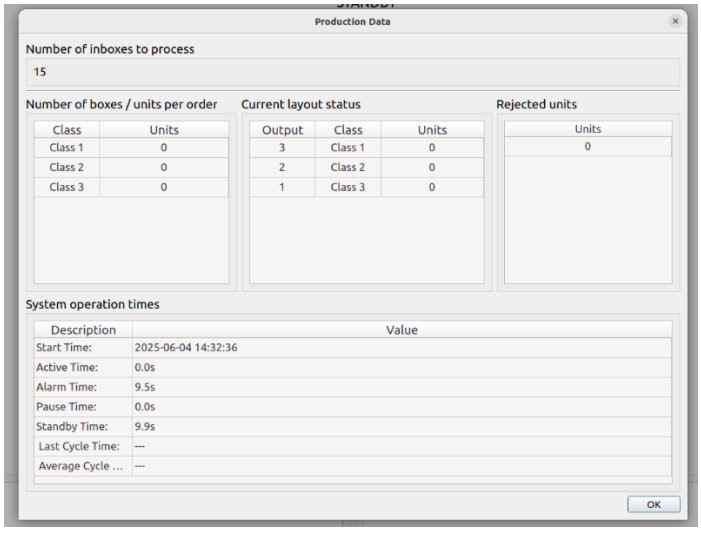


Figure 53. Order status monitoring in UC4

7 INFAR – UC5 – Fixtureless assembly in hand tool manufacturing sector

7.1 Use case overview

The INFAR use-case focuses on the ratchet wrench assembly. The main components to be assembled in the ratchet wrench are shown in Figure 54.

Funded by the European Union

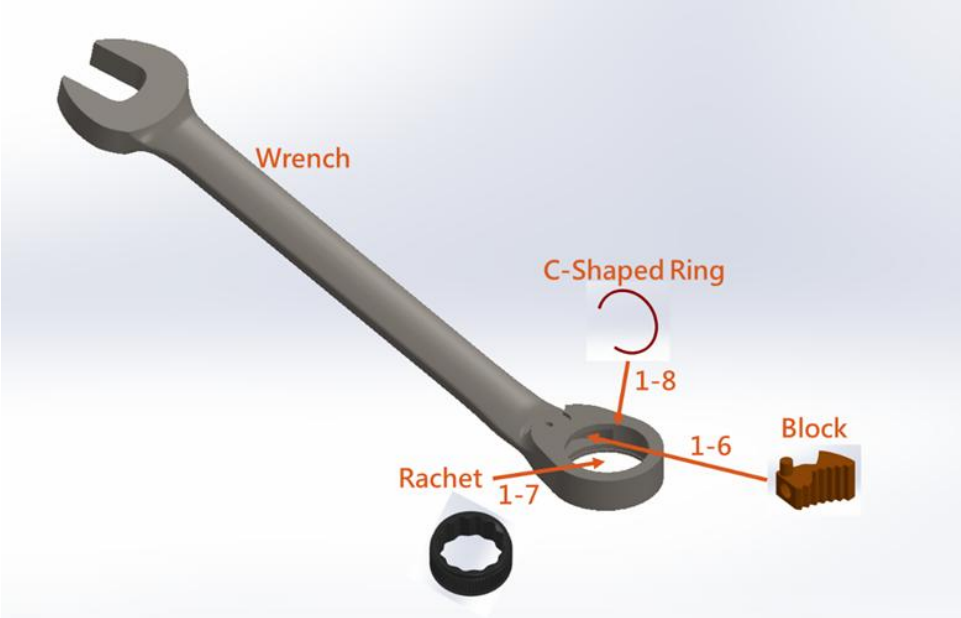


Figure 54. Main components of the wrench

Currently, operators assemble this ratchet wrench manually as shown in Figure 55.

Figure 55. Operators at the assembly tables

HARTU focuses on three main steps of the assembly process, as shown in Figure 56.

- Assembly step 6 – Block insertion
- Assembly step 7 – Ratchet insertion
- Assembly step 8 – C-shaped ring insertion

Funded by the European Union

Figure 56. Three main assembly steps: Step 6 (left), Step 7 (middle), Step 8 (right)

Automation of these assembly steps is challenging because of the small size of the parts that have to be manipulated.

7.2 Prototype description and components

A multi-robotic assembly system has been proposed for automating the assembly steps described in this use case. The demonstrator concept and the physical implementation is shown in next figures.

The main idea is to have the first robot arm to hold the main body of the ratchet wrench and the second robot to pick the components one by one following the assembly steps 6, 7, and 8. The assembly is performed via the coordination among the two robots. Before doing the assembly task, simulations on assembly steps are used to determine the best robot moving paths, configurations, coordination among robots, etc. as shown in Figure 57.

Figure 57. Simulated components for assembly steps

The demonstrator has been implemented following the simulation set-up shown in Figure 58.

Funded by the European Union

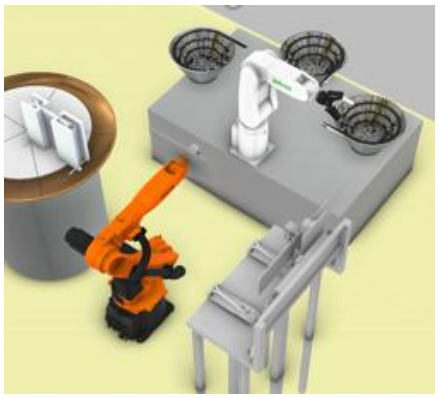


Figure 58. Simulated assembly process

The components included in the real demonstrator are the following:

- Two robots: AR605 or SJ605 (both are designed by ITRI)
- Specifically designed gripper to grasp the ratchet and the wrench. (Gripper is being designed by ITRI).
- Robot Cell Controller (eMIO designed by ITRI).
- Centralised robot coordination of the two robots with high-level commands, such as grasping points on the workpieces.
- Perception system, consisting of cameras at the ceiling or close to the robot end-effector to provide visual information for further object recognition, grasp planning, etc. FOVISION or SENSOPART products have been adopted.
- Multi-axis force/torque sensors mounted at the end-effectors to provide contact force information for assembly tasks.
- Workpiece loading/unloading mechanism for robot grasping of workpieces for next assembly movement (designed by ITRI).

Figure 59 shows the 3D layout and the AR605 of the demonstrator.

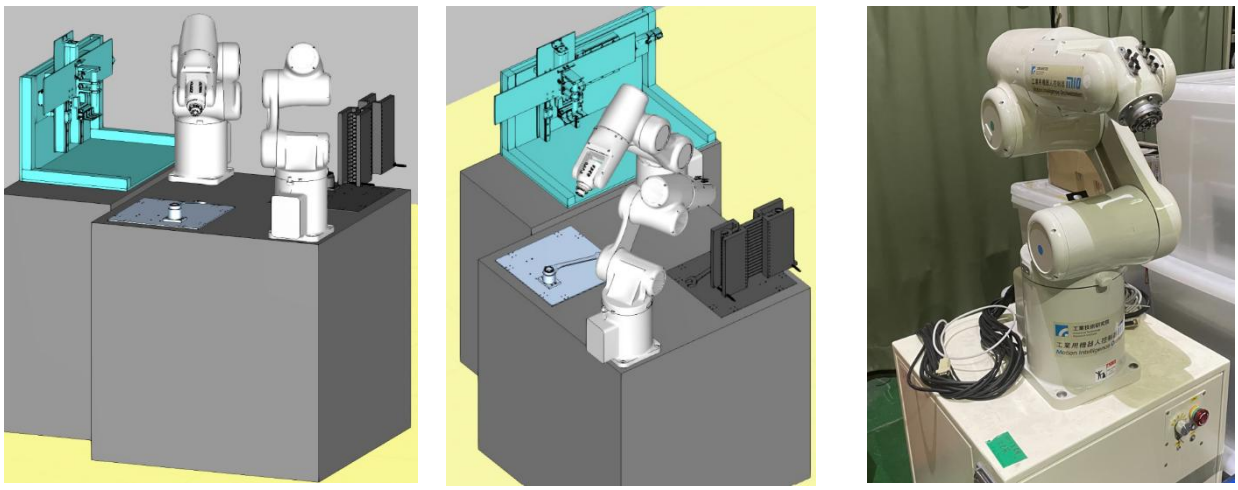


Figure 59. 3D layout (left) and the AR605 (right) of the demonstrator

The simulation for this demonstrator was used to verify possible solutions and designs, assembly procedures and algorithms.

Funded by the European Union

The sequence of actions in the demonstrator are:

1. Placing the main bodies of the ratchet wrench, the C-shaped rings, ratchets, and block-modules into the loading mechanisms.
2. The first robot picks a main body of the ratchet from the loading mechanism.
3. The second robot picks the block-module from the loading mechanism.
4. The assembly step 6 is performed by the second robot to insert the block-module into the main body of the ratchet wrench held by the first robot.
5. The second robot picks the ratchet from the loading mechanism.
6. The assembly step 7 is performed by the second robot to place the ratchet into the main body of the wrench held by the first robot.
7. The second robot picks the C-shaped ring from the loading mechanism.
8. The assembly step 8 is performed by the second robot to place the C-shaped ring around the ratchet.
9. Then, the first robot places the assembled ratchet wrench into a basket.

7.3 System control and GUI

ITRI designed a multi-robot assembly system to enable flexible, fixtureless manufacturing by leveraging coordinated robotic control, AI-driven learning mechanisms, and real-time sensor integration. This architecture supports intelligent manipulation, dynamic part alignment, and collaborative task execution without reliance on traditional physical jigs or fixtures.

The control architecture is illustrated in the figure below:

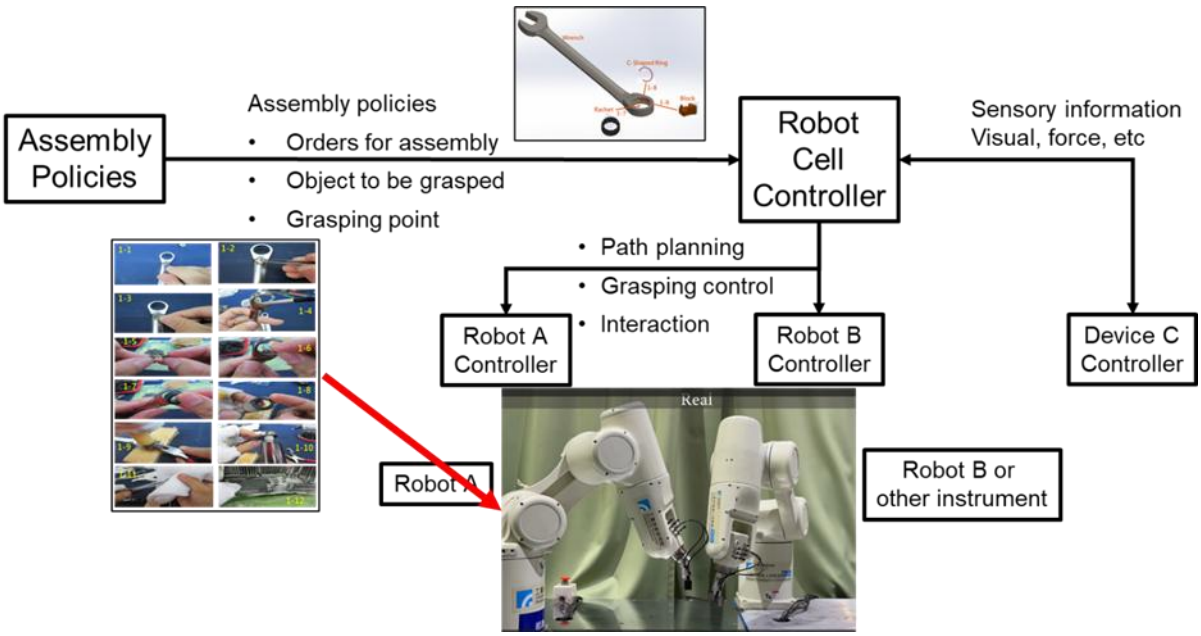


Figure 60. UC5 demonstrator control

At the core of this system lies the Robot Cell Controller, which functions as the central nervous system of the multi-robot virtual fixture-based assembly framework. It orchestrates the actions of all robotic units, ensuring seamless coordination across the entire cell. The controller is tightly integrated with virtual fixture software, sensor arrays, and vision systems. It continuously receives

Funded by the European Union

and processes real-time data from these sources to manage robot behaviors with high precision, according to the current state of the workpiece and assembly specifications.

The figure below illustrates the GUI for ratchet wrench assembly. In this figure, wrench size can be selected and the completeness of assembly can be checked automatically.

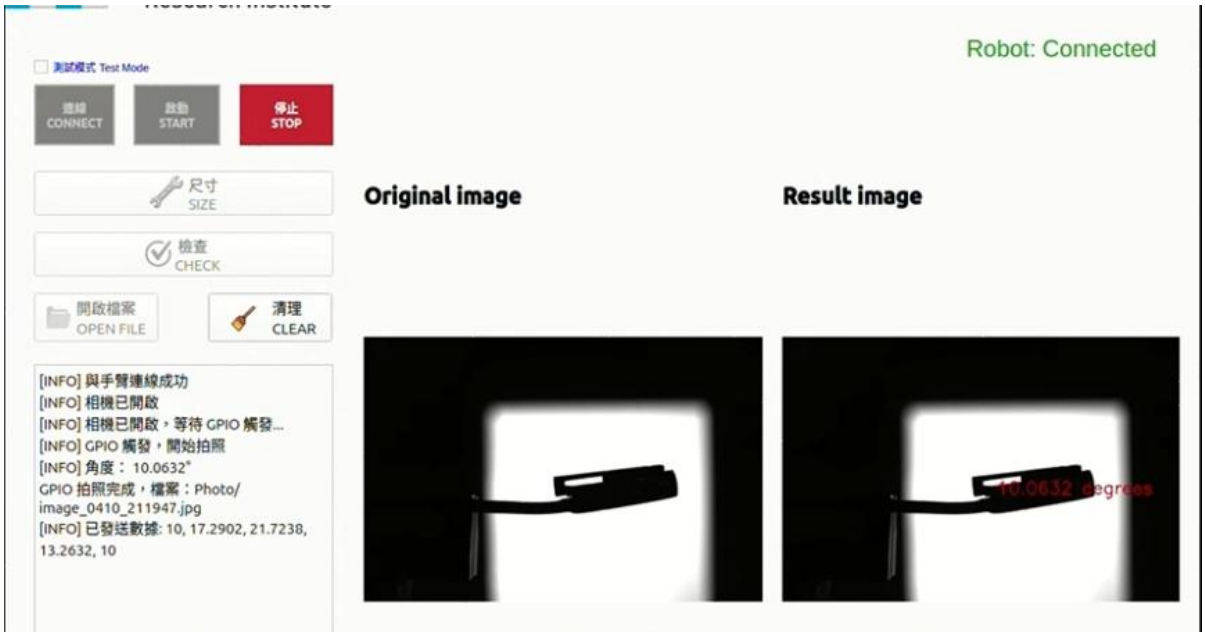


Figure 61. GUI for UC5

8 ULMA – UC6 – Order preparation: pallet to pallet

8.1 Use case overview

In logistics, there are different types of order preparation procedures depending on how the products arrive at the preparation area and how the orders are delivered.

- Input
  - Products arrive on pallets, this is mainly the case of bulky products packaged in carboards, large cans and sacks.
  - Small size products arrive in boxes, sorted or randomly distributed.
- Output
  - Products are stacked on pallets, either of one or multiple references.
  - Products are placed in boxes, sorted or unstacked (randomly).

This use case corresponds to the case in which products arrive on mono-reference pallets and are delivered on multi-reference pallets.

Funded by the European Union

In the order preparation area operators pick units from the incoming pallet (the one that has been transported from the warehouse) and place them on the pallets that will be finally delivered to the customer, as shown in Figure 62.

Some of the features of the use case are the following:

- The incoming pallets (Euro Pallet, EPAL) are always mono-reference and the output boxes are, usually, multi-reference.
- The warehouse management system informs the operator of the number of units that have to be picked from the incoming pallet. This information is available in a GUI and is displayed in a pick-to-light system.
- Operators manipulate the product by hand, and with the help of industrial manipulators for the heaviest products (they can weight up to 30 kg).
- In some few occasions, the incoming pallet transports a box with products inside, which must be manipulated individually to complete an order (e.g., to take a can from the box an put them on the output pallet).
- Operators use their own criteria to create the output pallet, trying to find the best combination to create stable pallets. For that, sometimes they move the already placed items and reposition them.
- .

Figure 62. Real example of output pallet at ULMA's customer

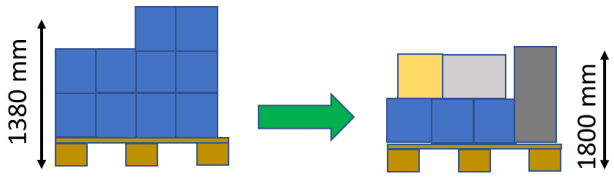


Figure 63. Pallet to pallet process

Figure 64. Example of multi-reference pallet

Two prototypes have been created: one at TEKNIKER for the validation of partial results and the second at ULMA. The main two differences among them are:

- The type of robot used.
- The way the input pallets are transported to the picking station (using a conveyor in the case of ULMA and manually in the case of TEKNIKER).

Both prototypes are described in the following sections.

Funded by the European Union

8.2 Prototype description and components at TEK

The prototype at TEK consist of the following elements:

- Robot KUKA KR210 R2700-2 /FLR.
- There is not a conveyor to transport the input pallets. Instead, they are moved by hand.
- 1 Photoneo L above the picking station (input pallet).
- 1 Mech Mind mounted on the robotic arm to monitor the status of the output pallet.
- 3 suction grippers with different configurations of vacuum grippers to handel boxes and cans of different sizes.
- A tool exchange station
- Pallets are moved by hand
- A control PC

The sequence of actions is like that of ULMA (see next section), except that the input pallets are moved by hand.

Figure 65. Lab setup at TEK

Funded by the European Union

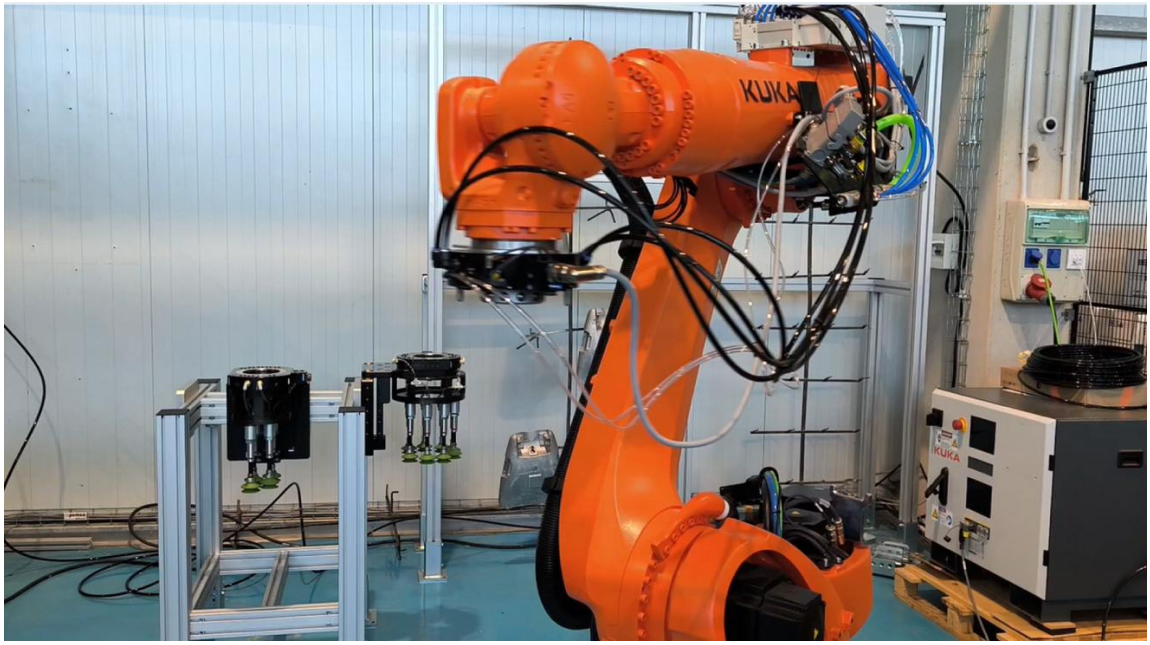


Figure 66. Tools and tool exchange station

8.3 Prototype description and components at ULMA

The demonstrator concept and the physical implementation is shown in Figure 67 and Figure 68:

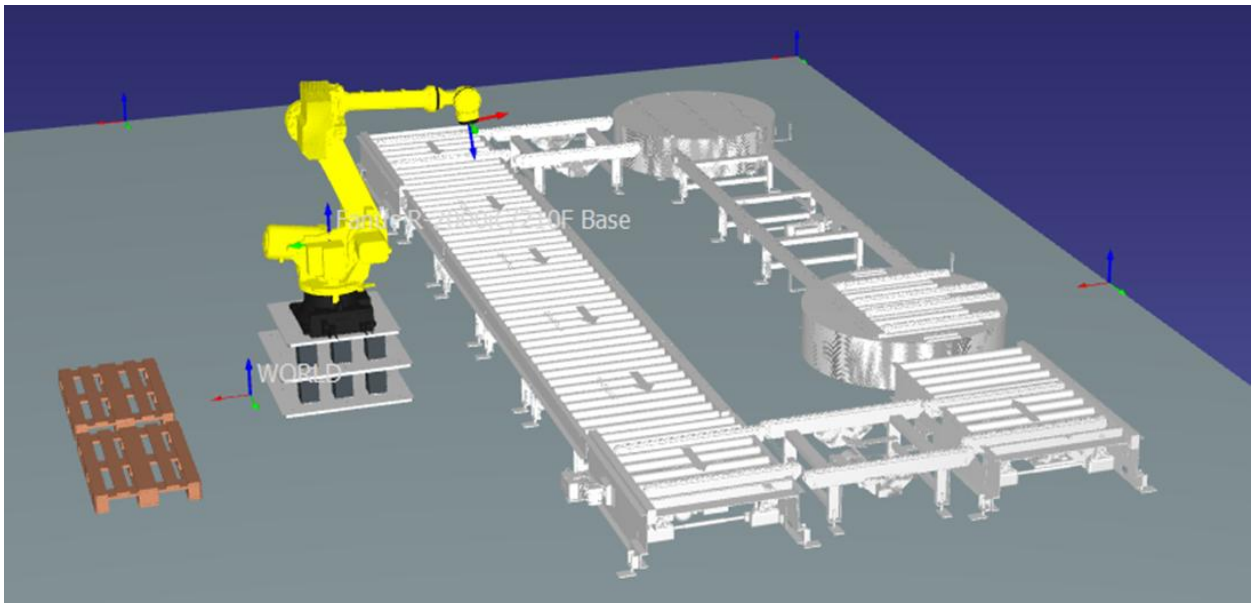


Figure 67. Design of the demonstrator at ULMA

The robot picks products from the pallets transported on the conveyor and creates a new multi-reference pallet (an order).

Funded by the European Union

Figure 68. Setup at ULMA

The components included in the demonstrator are the same than in the prototype at TEKNIKER with the following differences:

- Robot FANUC R-2000Ic/210F.
- Conveyor for transporting the input pallets (circular path).

The sequence of actions in the demonstrator are:

1. The system calculates the mosaic (position of each part) to be created for an order.
2. The warehouse management system delivers the input pallets based on the position of the products in the mosaic.
3. For each pallet arriving at the picking station:
    a) An image of the input pallet is captured with the fixed camera.
    b) The system decides which product has to be picked.
    c) The robot picks the product.
    d) The robot places the product in the corresponding position of the mosaic.
    e) The robot takes an image of the mosaic with the embedded camera.

Funded by the European Union

If there is a mismatch with respect to the proposed mosaic, it stops and an alarm is generated (light or message) to inform the operator.

f) The sequence is repeated for the number of items to be picked. When finished, the pallet leaves the picking station and a new one arrives.

Figure 69. First integration tests for UC6

Funded by the European Union

8.4 System control and GUI

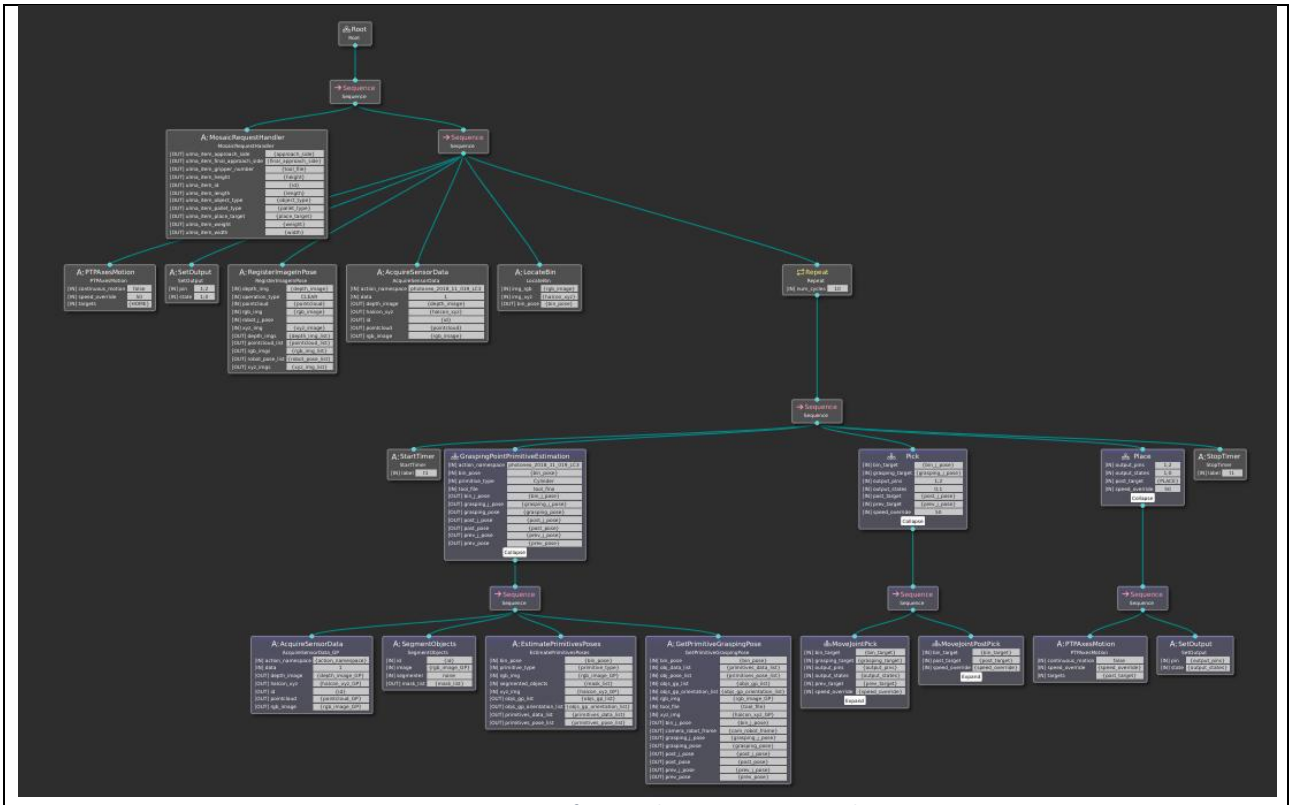


Figure 70. BT for UC6 demonstrator control

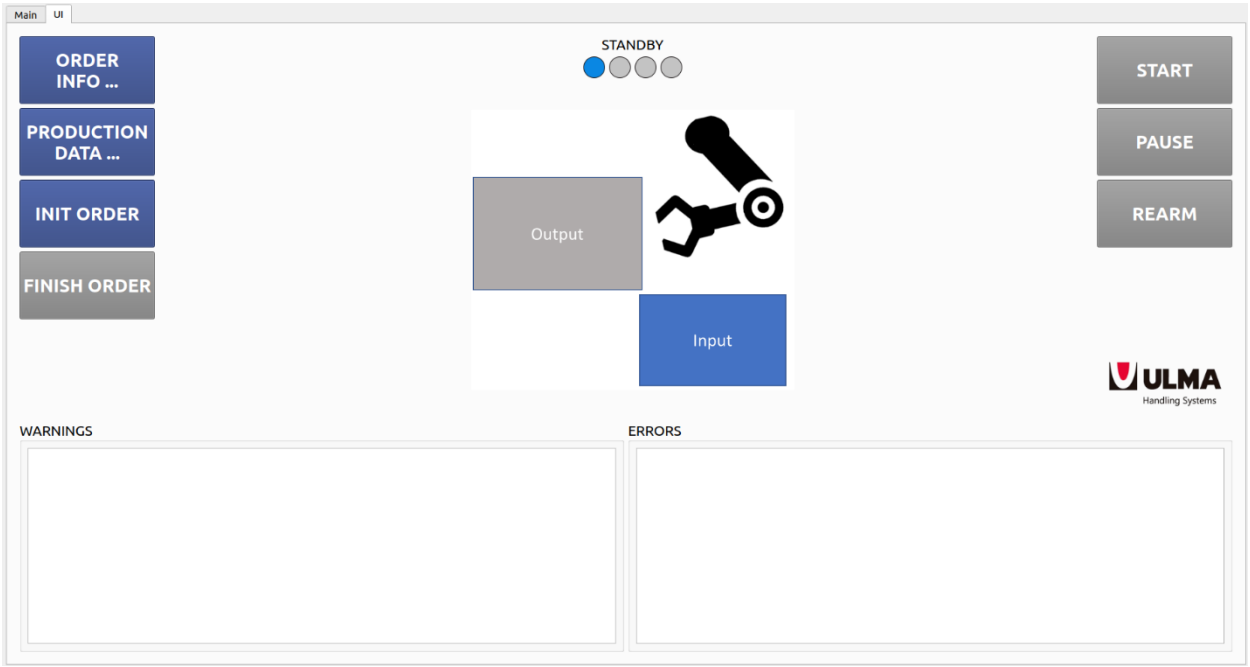


Figure 71. GUI for UC6

Funded by the European Union

9 ULMA – UC7 – Order preparation: box to box

9.1 Use case overview

These use case corresponds to order preparation in logistics centres, where products arrive in boxes, are picked manually by operators and placed in multi-reference boxes to complete an order.

The products are placed unstacked in the output box because they are then transported to a workstation to be packed in the final packaging in which they will be delivered to the customer.

Some of the features of the use case are the following:

- Products come in a huge variety of shapes, materials, and dimensions.
- Humans use their both hands and sometimes they pick more than one product at once.
- Operators are informed through the pick-to-light system on the number of items that have to be picked and the destination box.

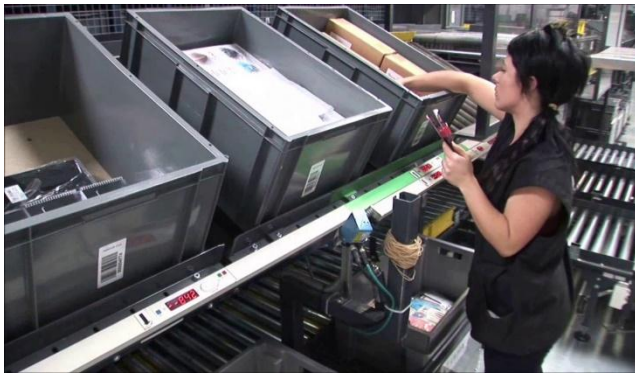


Figure 72. Manual picking

Figure 73. Placing the products in the output box

The concept proposed is presented in the following figure. It consists of a robot picking items from one box and dropping them in the output box which is used to create an order. The boxes are automatically transported from/to the warehouse, an advanced perception system identifies the position of the parts, and the robot automatically generates the trajectory for both picking and releasing operations (HARTU results).

Depending on the production required, it is possible to have one or more robots, either in a fixed position or mounted on a linear track. Similarly, the number of input and output boxes managed can vary. The cameras on top of the input boxes can be fixed or mounted on a rail.

Funded by the European Union

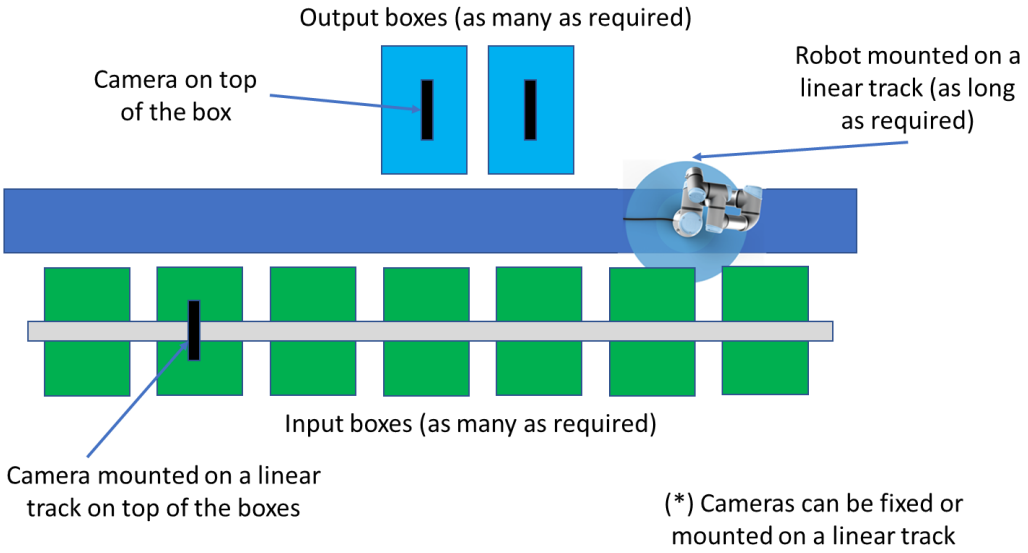


Figure 74. Design of the box to box order preparation

This prototype has been setup and will be validated at TEK.

9.2 Prototype description and components

As a proof of concept only the preparation of one output box (one order) at a time has been implemented, and due to space limitations, up to 5 input boxes will be managed (this means that in this prototype an order can only be composed of a maximum of 5 different product references, but many items of each are possible).

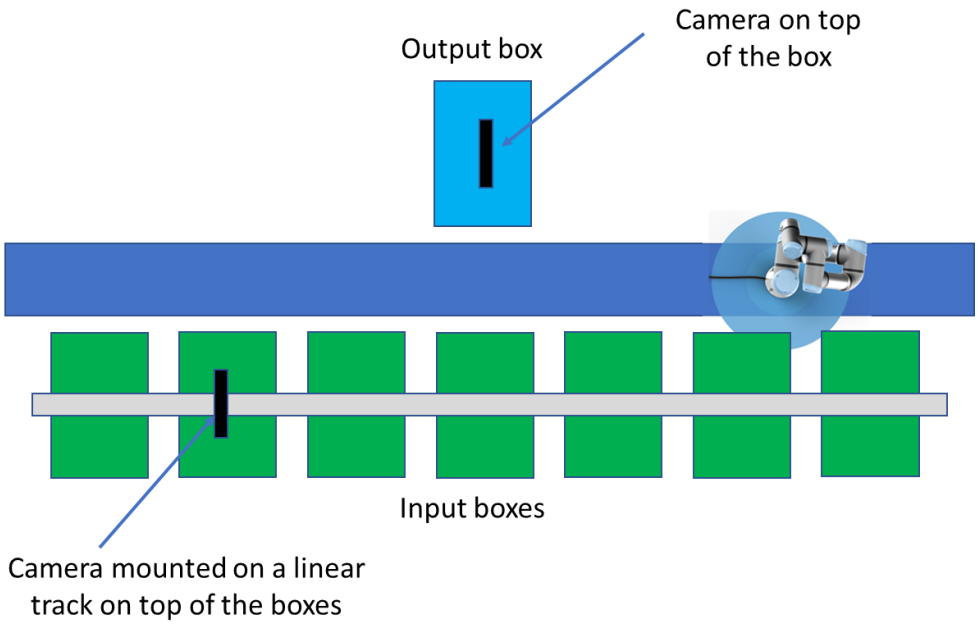


Figure 75. Design of the demonstrator at TEK

Funded by the European Union

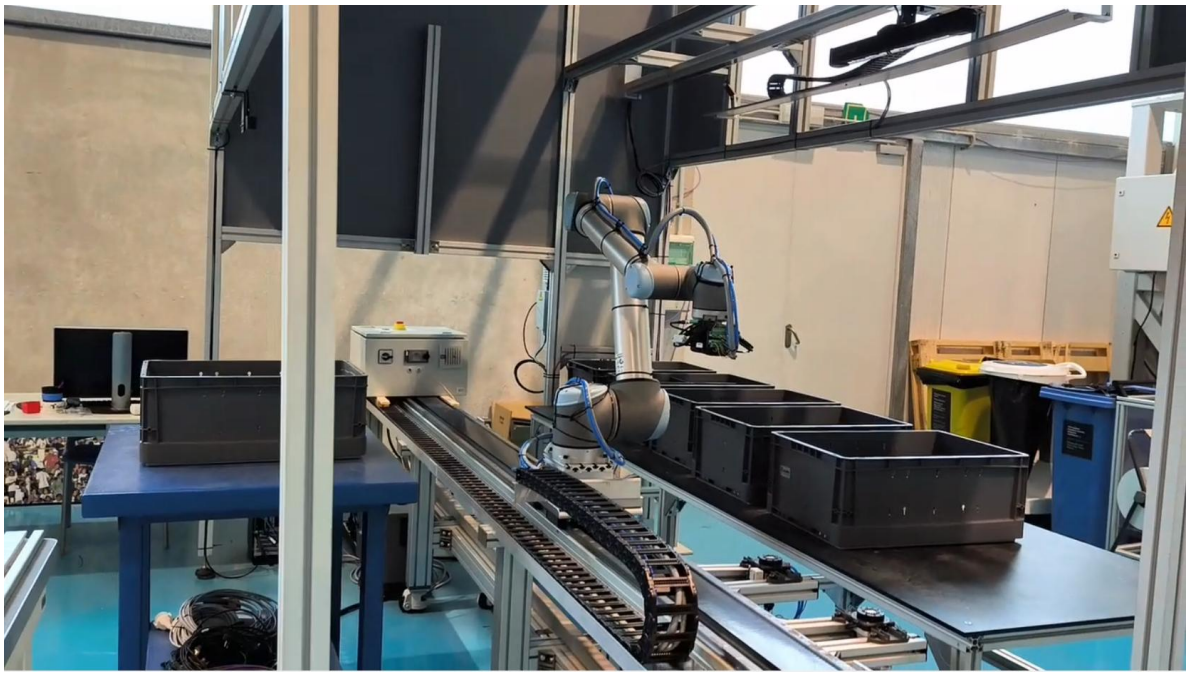


Figure 76. Setup of the demonstrator at TEK

The components included in the demonstrator are the following:

- Robot: UR 10 mounted on a 4 meters long linear axis.
- Plastic boxes for product delivery and order preparation.
    - Up to 5 input boxes (single reference): bulk products.
    - 1 output box (multi-reference) to prepare an order.
- Cameras
    - Photoneo XL mounted on a linear axis to monitor up to 7 the input boxes.
    - ZED2i camera to monitor the output box.
- Grippers
    - 3-finger gripper for products with cylindrical geometries that cannot be gripped by suction.
    - Suction gripper for products with a suitable surface, e.g., boxes.
- Tool change station
    - The tools are equipped with quick-change devices.

Funded by the European Union

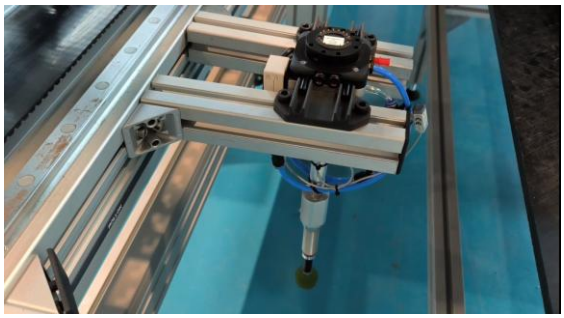


Figure 77. Suction gripper in the tool station

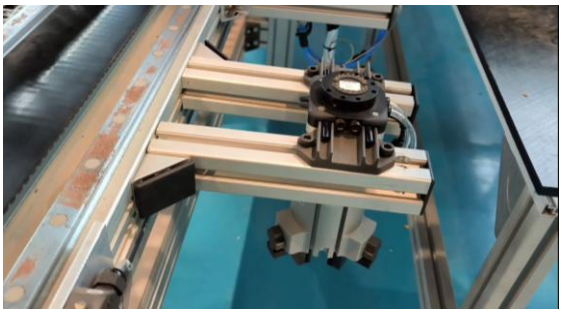


Figure 78. 3 Finger gripper in the tool station

The sequence of actions in the demonstrator are:

1. The warehouse management system sends the list of items to complete an order
2. The warehouse management system delivers the input boxes to complete the order
    a) In the LAB version the boxes are placed on a table and their position is sent to the robot
3. For each product to complete the order:
    a) In the LAB version, the robot moves to the input box position.
    a) The robot changes the required tool.
    b) An image of the input box is captured with the camera mounted on the linear axis.
    c) The system decides which product will be picked.
    d) The robot picks the product.
    a) The robot releases the product into the output box (order), trying not to create piles.

    Sometimes, linear translation of the robot to the position of the output box is required.
    b) An image of the output box is taken for the next iteration.

    This information is used to ensure that products are not stacked.

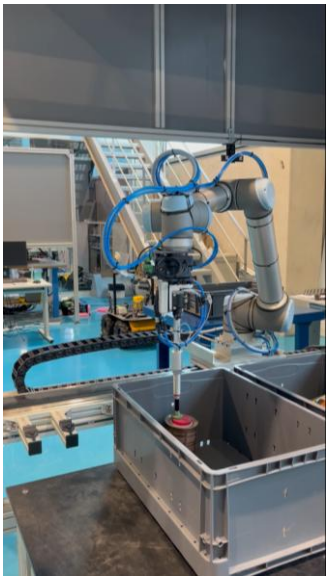


Figure 79. First integration tests for UC7

Funded by the European Union

9.3 System control and GUI

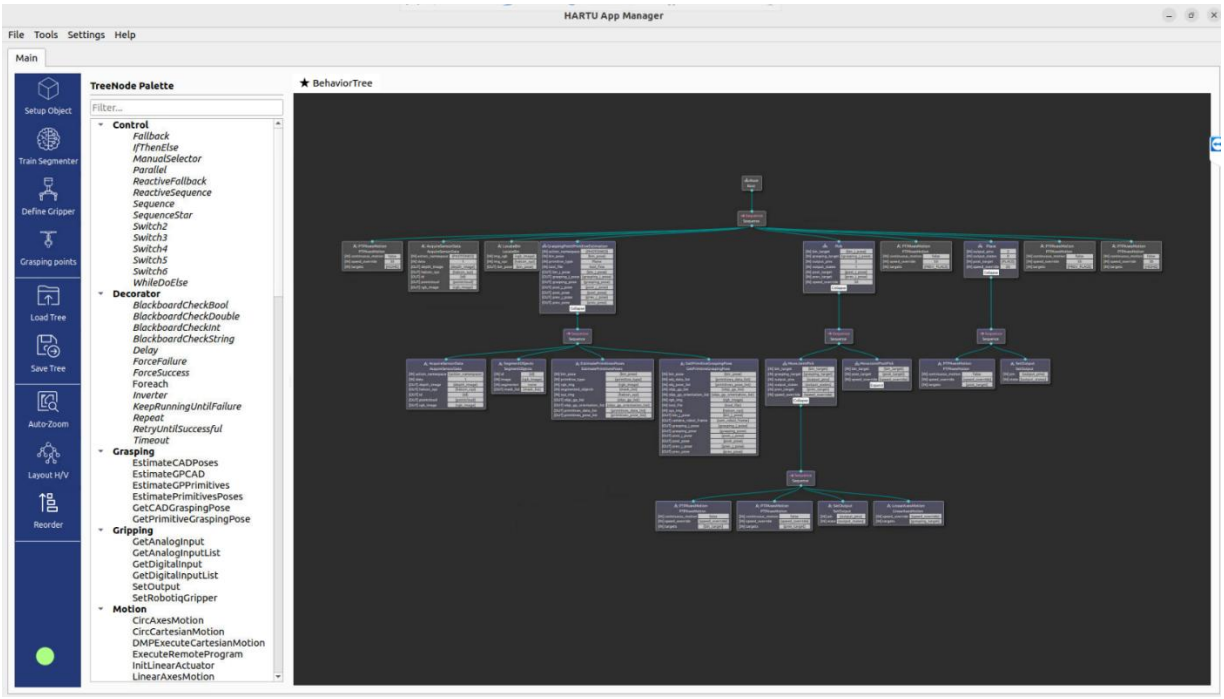


Figure 80. BT for UC7 demonstrator control

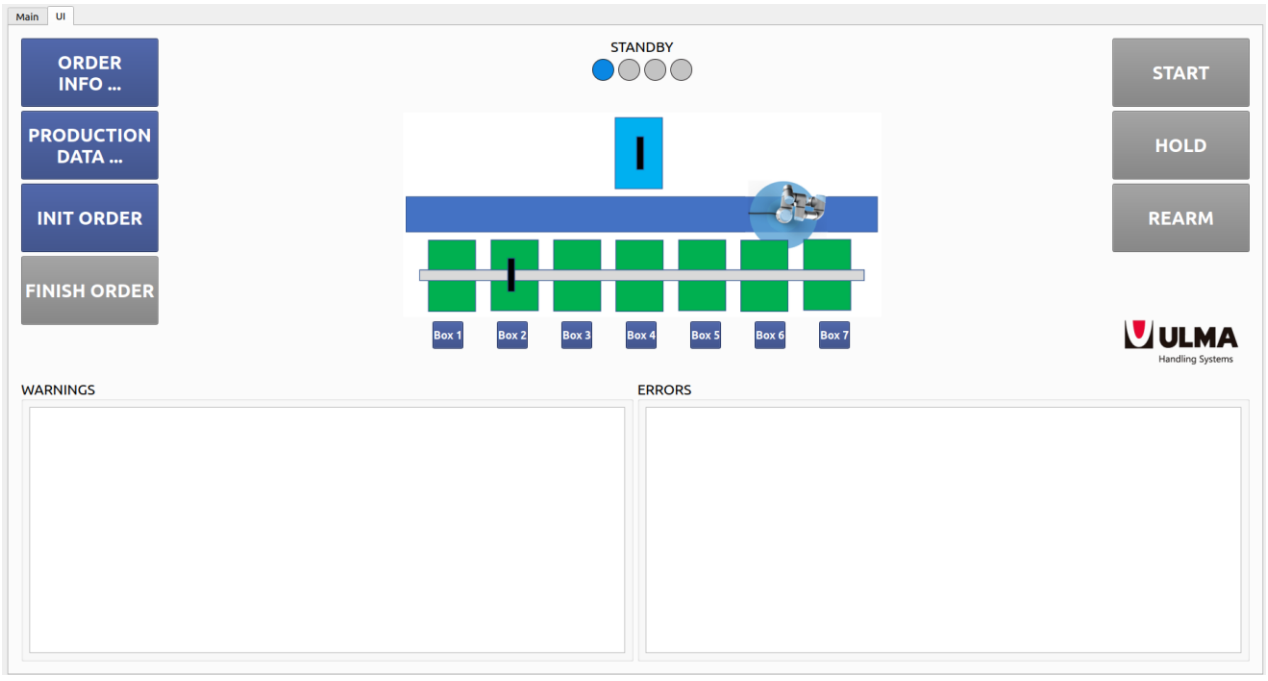


Figure 81. GUI for UC7

Funded by the European Union

10 HARTU Reference Architecture. Update

10.1 Introduction

Building on the iterative and incremental methodology outlined in the Conclusions section of D2.1, this document provides an updated account of the refinements and improvements applied to the overall Reference Architecture (RA) of the HARTU Project. These revisions are part of a continuous process to enhance the software architecture, communication mechanisms, and technical design, ensuring alignment with evolving project needs. By adopting a more detailed and lower-level approach, the updated architectural style emphasizes the **Component Pipeline** that facilitates interaction between humans and machines. This shift in the architectural design approach, mutually agreed upon by the project's technical partners, aims to increase the transparency of the solution development. Specifically, it offers a more technical perspective on refinements and improvements, ensuring a clearer representation of the software structure and its components. This approach reflects a more consolidated understanding of all software components and their interactions, an understanding that has matured during this second iteration of development.

10.2 Key updates

Key updates include:

- **Enhanced Technical Design**: The introduction of a lower-level architectural framework better delineates the core software components and their respective roles in human-machine interaction.
- **Refined Communication Mechanisms**: Adjustments to the communication strategies among system components aim to optimize efficiency and robustness in real-time operations.
- **Improved Transparency**: A more technical presentation of the updates ensures that stakeholders can better track changes and align their contributions with the overall project goals.

This iteration not only builds upon the insights gained during the first phase but also lays a strong foundation for subsequent development cycles, fostering greater collaboration and understanding among the technical teams involved.

The definition process which consisted of collaborative activities such as those carried out during the architecture review meetings, is presented in the Figure 82:

Funded by the European Union

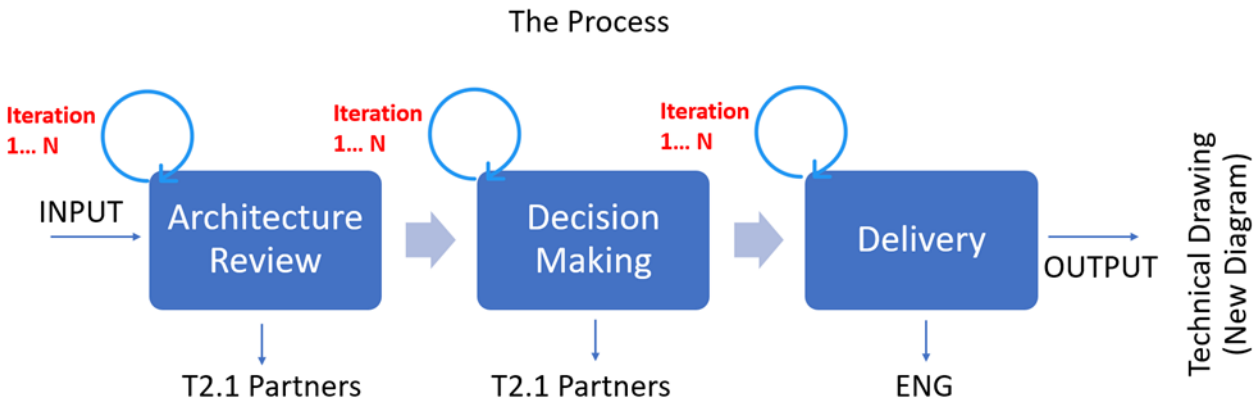


Figure 82: HARTU RA - New Model definition Process

Iterative cycles have been envisaged for the new design, for the decision-making process and for the delivery of the different versions of the new model. The output, refined through multiple rounds of discussion and feedback, was ultimately consolidated into the final design (see Figure 83).

The ability to seamlessly transition between architectural patterns, each emphasizing the diverse services offered by the HARTU solution for pilots, underscores a critical competitive advantage: the capacity to deliver solutions on-demand, quickly and securely. By implementing a robust code pipeline to automate both applications and infrastructure, operational speed has been significantly improved, enhancing code quality, and simultaneously empowering developers and system integrators with greater autonomy while ensuring system maintainability.

The principle of **Continuous Delivery**—defined as "the ability to deliver changes of all kinds into the hands of users quickly, safely, and sustainably"—serves as a cornerstone of this approach. Positioned between **Continuous Integration and Continuous Deployment**, HARTU leverages the advantages of a CI/CD environment to streamline both development and release processes. This integration fosters agility, reliability, and efficiency, ensuring that changes are delivered in alignment with user needs and system goals.

The Component Pipeline architectural style provides a framework for emphasizing key characteristics of the system, organizing system components into a sequence of interconnected processing elements, commonly referred to as "components". Each component in the pipeline performs a specific operation on data and passes its output to the next component in the sequence. This style is particularly suited for systems where data transformation, processing, or flow control is essential.

Key characteristics of this new approach are:

- **Sequential Data Flow**: Data flows through a defined sequence of processing steps. The output of one component becomes the input for the next, creating a streamlined process.
- **Parallel or Time-Sliced Execution**: Pipeline components can be executed simultaneously (parallel processing) or in time-sliced intervals, depending on system requirements. This enhances system performance by reducing bottlenecks.

Funded by the European Union

- **Modularity and Reusability**: Each component is implemented as an independent component, allowing flexibility in design and easy replacement or reuse of components.
- **Scalability**: New components can be added to the pipeline to handle additional functionalities without disrupting the existing system. The architecture supports horizontal scaling by duplicating components across multiple processing units.
- **Transparency**: The pipeline clearly illustrates the flow of data and the responsibilities of each component, making the architecture easier to understand, debug, and maintain.

It is necessary to highlight how, even though the approach to the design has been changed, some significant architectural principles have been maintained compared to the first iteration (D2.1), such as the modularity of the solution and the scalability.

The main advantages of using this architecture style are enhanced performance through parallel execution for faster data processing, simplified maintenance by clearly separating concerns to prevent updates from affecting other components and facilitated debugging with a linear flow that makes it easier to trace errors or inefficiencies.

Within this context, the HARTU partners collectively decided to focus on showcasing the communication pathways between the primary software components of the system. These components play a critical role in enabling effective human-machine interaction across all HARTU scenarios. By highlighting each component, its corresponding functionality becomes apparent. For example, this approach clarifies the system capabilities in areas such as **Object Perception, Grasp and Release Planning,** and **Robot Application Planning and Control**. Moreover, this method not only provides insight into the system architecture but also helps underscore its practical applications and the interconnectivity required to support seamless operations.

Funded by the European Union

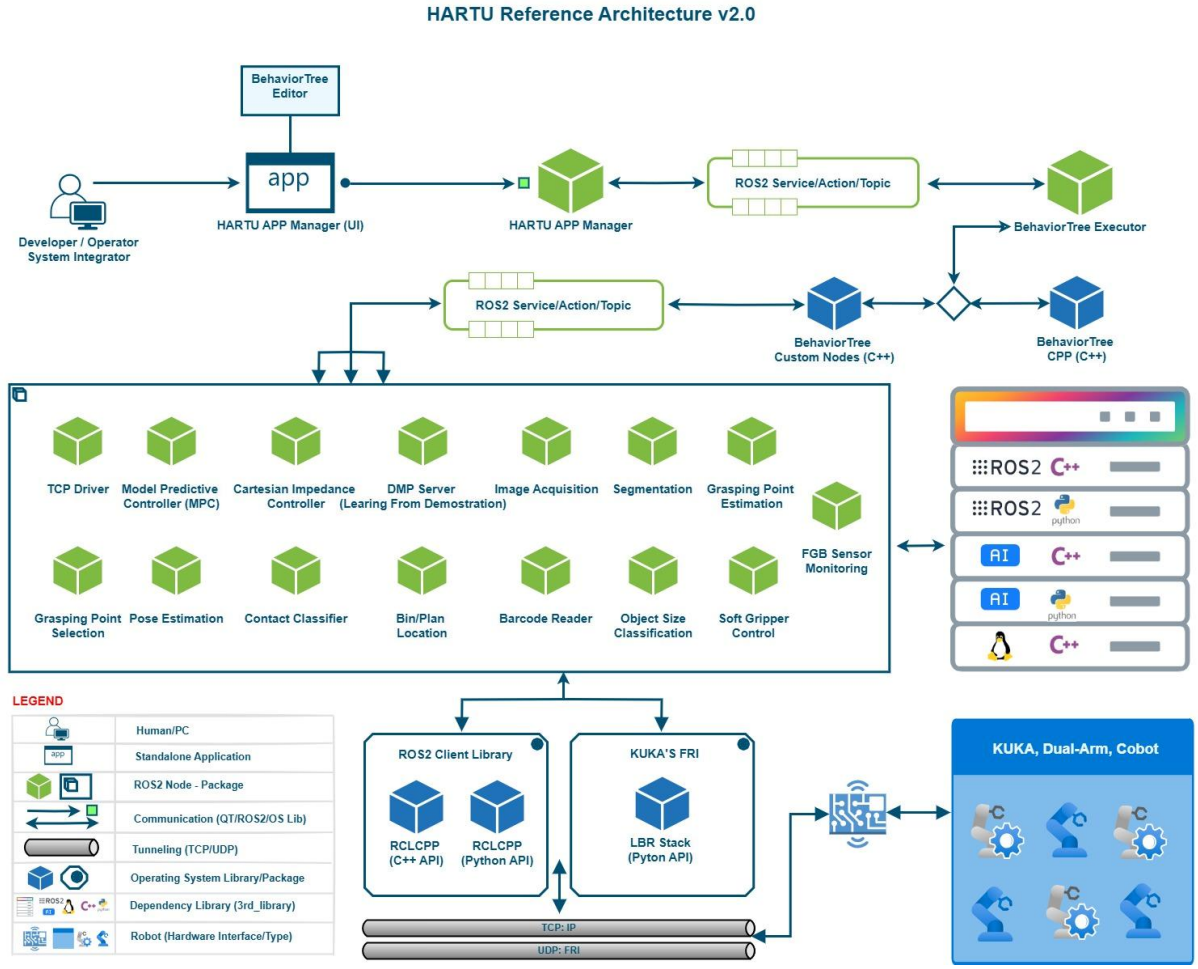


Figure 83: New HARTU RA Reference Model (v2 - FINAL)

The new **Reference Architecture model**, based on the Component Pipeline, introduces a reimagined design for the HARTU software architecture. This model illustrates a sequence of interconnected processing elements, the ROS2 nodes, where the output of one element serves as the input for the next. These elements are interconnected sequentially but can also be executed in parallel through a pipeline created with the App Manager. This mode allows the pipeline elements to work on different data at the same time or through time-sliced operations, enhancing efficiency and responsiveness.

The architectural diagram illustrates the components that contribute to the execution of one or more system functions. It is designed to be interpreted bidirectionally—from top to bottom or vice versa—offering a comprehensive and aggregated view of the system capabilities, both in terms of functionality and potential.

Below is provided a detailed breakdown of the **operational flow**, describing the specific functions of each individual element within the solution. This step-by-step explanation, organized from a top-to-bottom perspective, clarifies the interactions and dependencies between components, offering a comprehensive understanding of how these elements collectively enable the seamless execution of system processes.

Funded by the European Union

The HARTU system provides a modular software architecture designed to enable system integrators and operators to develop and execute robotic manipulation applications efficiently. Users interact with the system using the **HARTU App Manager** for defining objects to manipulate, configuring a virtual scene, generating image datasets for segmentation training, specifying gripper designs, and setting grasp points for gripper-object combinations.

The **HARTU App Manager** offers also a centralized graphical user interface. This application enables users to create new robotic applications or execute existing ones, acting as a one-stop solution for accessing all functionalities required for robotic manipulation. The HARTU App Manager is implemented as a ROS2 node, comprising two main parts:

- a user interface (**HARTU App Manager (UI)**), built with Qt5[1],
- and a backend (**HARTU App Manager**), which handles application logic.

Communication between the UI and the backend is facilitated via Qt's Signal/Slot mechanism. The Signal/Slot mechanism in Qt is a type-safe, event-driven communication system, where signals emitted by one object can be automatically connected to slots (functions) in another object, ensuring seamless and efficient interaction between the UI and backend components of the HARTU App Manager.

The UI also integrates a **BehaviorTree Editor**, allowing users to design and edit trees through an intuitive drag-and-drop interface. These trees are saved in an XML format, which serves as a standardized and portable representation of task logic. They form the core of the decision-making processes for robotic tasks by defining the **sequence**, **conditions**, **and actions** to be executed. This structure seamlessly connects with the BehaviorTreeExecutor component, described in the following paragraph, which manages the lifecycle of these trees and bridges task planning with robotic execution.

The execution of Behavior Trees is managed by the **BehaviorTreeExecutor** component, which provides ROS2 services to control the lifecycle of trees, including **creation, execution, and stopping**.

The HARTU App Manager leverages these services to execute trees, enabling seamless integration between task planning and robotic execution. Built using the *BehaviorTree.CPP v3* library[2], BehaviorTreeExecutor supports a highly modular and dynamic approach, utilizing a plugin-based architecture to extend functionality.

This plugin mechanism allows the system to load **Custom Behavior Tree nodes** tailored to specific robotic tasks. These nodes provide capabilities such as sensor data acquisition, pose estimation, grasp point calculation, trajectory planning, and point-to-point motion, among others. Each node is implemented in C++ using *BehaviorTree.CPP v3* and designed to integrate dynamically within Behavior Trees. They communicate with other **ROS2 nodes** via **actions**, **services**, and **topics**, ensuring a standardized and interoperable approach to robotic task execution. The modularity achieved by the implementation of Custom Behavior Tree nodes and ROS2 standardized messages not only allows the user to mix-and-match nodes to tailor a solution for their robotic application. It

[1] https://doc.qt.io/qt-5/
[2] https://www.behaviortree.dev/docs/3.8/intro

Funded by the European Union

also facilitates developers to continuously improve, integrate, and deploy their solutions in the HARTU App Manager.

ROS2 nodes providing these specific functionalities are implemented in either C++ or Python, using the respective ***rclcpp*** and ***rclpy*** client libraries[3]. These libraries enable robust and efficient communication, supporting features such as real-time execution in C++ and rapid prototyping in Python. By standardizing message exchange through ROS2 interfaces[4], these nodes ensure seamless interaction within distributed robotic systems.

To enhance functionality, these ROS2 nodes **leverage Third-Party Libraries** (depicted as **Dependencies Libraries** in Figure 83 – see 'LEGEND' – specialized in areas such as **computer vision, motion planning, and geometric computations**. This allows for the efficient and modular implementation of advanced capabilities like object detection, pose estimation, trajectory generation, and grasp point calculations.

To meet the diverse needs of robotic applications, the **communication architecture** employs a hybrid approach by using the **TCP** and **UDP** protocols in addition to the standard ROS2 Data Distribution Service (DDS) middleware. Standard robotic movements, such as point-to-point, linear, or circular trajectories, are handled via TCP with custom message definitions for reliable and robust communication. For applications requiring low-latency data exchange and joint torque-based control interfaces, such as force reproduction in the *Learning from Demonstration* approach or *Adaptive Model Predictive Control*, UDP communication is employed using KUKA's Fast Robot Interface (FRI). This allows real-time cyclic data exchange at millisecond intervals, critical for precise and responsive robotic control, and on this basis provides extended control modes in addition to joint position control. To use these features, a dedicated ROS2 hardware interface and controller (in terms of the ROS2 control standard API) is employed through the package "LBR-Stack".

This **multi-protocol communication** strategy ensures that the HARTU system is **adaptable to robots from multiple manufacturers**. Each robot requires a specific program running on its controller to manage communication, enabling seamless integration and interoperability. This flexibility ensures the architecture applicability across a wide range of robotic platforms and operational scenarios.

10.3 ROS2 nodes components

Below is a description of each ROS2 Node visible in the architectural model:

- **TCP Driver**:
  - *General Description*: this component is responsible for establishing and managing reliable communication between the HARTU system and robotic controllers using the TCP protocol. It ensures the execution of standard robotic movements, such as point-to-point, linear, or circular trajectories, by processing custom message definitions and translating them into commands understood by the robot.
  - *IN/OUT*:

[3] https://docs.ros.org/en/foxy/Concepts/About-ROS-2-Client-Libraries.html
[4] https://docs.ros.org/en/foxy/Concepts/About-ROS-Interfaces.html

Funded by the European Union

        - Input: Custom message definitions (from the HARTU system or other control components).
        - Output: Commands understood by the robotic controllers, enabling the execution of movements.
    - *Interaction with other ROS2 Nodes*: The TCP Driver communicates with other ROS2 nodes to relay commands to robotic controllers. It processes messages from nodes that define movement parameters and translates them into TCP packets that are sent to the robots, facilitating movement execution.
- **Model Predictive Controller (MPC)**:
    - *General description*: this component utilizes model-based optimization to track a state reference trajectory in tasks that involve contact with the environment. The novelty of this controller lies in its handling of uncertainty in assembly tasks. Gaussian Mixture Models (GMM) are employed to adapt the MPC model according to expected environmental parameters. A set of pre-trained GMMs serves as the robot's task knowledge. When the task begins, the controller selects an appropriate task model based on the GMMs' likelihoods. The uncertainty from the chosen GMM is then used to adjust the MPC cost function weights, allowing the controller to regulate compliance and stiffness depending on the confidence in the model.
      The system is divided into two main components:
        - *MPC Node*: The controller itself, responsible for computing the control signals.
        - *GMM Node*: Provides the MPC with the necessary task-related knowledge and uncertainty information.
    - *IN/OUT*:
        - Inputs to GMM Node:
            - Predefined Knowledge for Tasks:
                - Pre-trained **GMMs** (Gaussian Mixture Models).
                - Pre-trained **uncertainty models**.
            - Optional Generic Parameters (for online updates):
                - Online "improved" uncertainties from the **Contact Classifier** (used to update predefined models).
        - Outputs from GMM Node to MPC Node:
            - Selected GMM for the current task.
            - Environmental parameters expectation (e.g., mass-inertia, friction).
            - Model uncertainty (for task model adaptation).
        - Additional Inputs to MPC (from DMP Node or Cartesian Impedance Controller, optionally):
            - Reference pose
            - Reference force
        - Outputs from MPC Node:
            - Joint torques generated using Inverse Kinematics (IK), which are then used to command the robot's joints.
    - *Interaction with other ROS2 Nodes*:

Funded by the European Union

- The GMM Node communicates with the MPC Node by providing it with the relevant task model, environmental parameters, and model uncertainty.
- The MPC Node also interacts with the DMP Node or Cartesian Impedance Controller to receive reference poses and forces that guide the task execution.
- The MPC Node uses Inverse Kinematics (IK) to compute joint torques for robot control.

- **Cartesian Impedance Controller**:
  - *General Description*: this component is designed to achieve compliant robotic behaviour through real-time control of joint torques. It modulates stiffness and damping parameters in Cartesian space, allowing the robot to adapt its movements based on task requirements and external interactions. The controller integrates the RBDL Library for kinematic and dynamic computations, leveraging the robot's URDF (Unified Robot Description Format) model to accurately represent the robot's structure and dynamics. These computations inform the Cartesian Impedance Control Law, which generates joint torques based on the desired Cartesian pose, desired external forces, and system dynamics. The system allows for compliant behaviour by adjusting stiffness and damping parameters, making the robot responsive to environmental interactions. Additionally, null space trajectories in joint space are used to optimize secondary objectives, such as avoiding joint limits or improving manipulability.
  - IN/OUT:
    - Input to the controller:
      - *Desired External Forces (Cartesian Wrench)*: specifies the force the robot should exert in Cartesian space for tasks like force-controlled assembly or surface interaction.
      - *Cartesian Pose*: defines the desired position and orientation of the end-effector in space.
      - *Stiffness and Damping Parameters*: Determines the compliance characteristics of the robot, adjusting how rigidly or flexibly it responds to positional errors and external forces.
      - *Null space Trajectory in Joint Space*: Provides additional degrees of freedom to optimize the robot's joint configurations for secondary tasks (e.g., avoiding singularities or joint limits).
      - *URDF (Unified Robot Description Format)*: Supplies the robot's kinematic and dynamic model for accurate force, torque, and motion planning computations.
    - Outputs from the Controller:
      - *Joint Torques*: The calculated torques applied to the robot's joints to achieve the desired Cartesian motion and force application while maintaining compliance

Funded by the European Union

- *Robot State Feedback*: Real-time feedback of the robot's current joint positions, velocities, and other state variables, used for closed-loop control to ensure accurate and stable performance.

  - *Interaction with other ROS2 Nodes*:
    - The Cartesian Impedance Controller interacts with the RBDL Library for performing kinematic and dynamic computations, which are essential for generating accurate joint torques.
    - The URDF is used to model the robot's kinematics and dynamics, feeding into the control law to adjust the robot's behaviour.
    - The controller's outputs (joint torques and robot state feedback) are fed into the Robot Control System, allowing the robot to perform tasks like force-controlled assembly or surface interaction while maintaining compliance.

- **DMP Server (Learning from demonstration)**:
  - *General Description*: this component has two primary modes: demonstration and learning, and execution of learned DMPs. In the demonstration phase, the system captures user demonstrations of assembly tasks, recording motion trajectories and force data. This data is processed to learn **Dynamic Movement Primitives (DMPs)**, which provide a flexible representation of the demonstrated movements. In the execution phase, the learned DMPs are used to perform similar tasks, adapting them to new start and goal positions while maintaining compliance through motion and force control. The system ensures that the robot behaves flexibly and safely through compliant control during execution, adapting to changes in the environment and task requirements.
    During the demonstration phase:
    - The Data Recorder captures joint states, Cartesian poses, and interaction forces.
    - The Dual DMP Controller processes this data and learns the movement patterns, encoding them into DMPs that capture both spatial and temporal characteristics of the task.

    In the execution phase:
    - The Dual DMP Controller retrieves the stored DMPs and adapts them to new conditions.
    - The adapted poses are sent to the Compliant Control components for the robot's arms.

  - *IN/OUT*:
    - Inputs to the DMP Server:
      - *User Demonstrations (for learning phase)*: motion trajectories and force data captured during task demonstration.
      - *Stored DMPs (for execution phase)*: Pre-learned Dynamic Movement Primitives for task execution.

Funded by the European Union

- *New Start and Goal Positions (for execution phase)*: Information required to adapt the learned DMPs to new conditions.

▪ Outputs from the DMP Server:

- *Adapted Target Poses*: poses generated from the learned DMPs for task execution.
- *Real-Time Feedback*: continuous feedback about the robot's joint states and overall behaviour.

○ *Interaction with other ROS2 Nodes*:

▪ The Data Recorder needs to obtain the robot state as well as force torque measurements from the state publisher components to capture the robot's joint states, Cartesian poses, and interaction forces, which is used for learning DMPs.

▪

▪ During execution, the DMP Controller outputs the adapted trajectories to the robot controllers such as for compliant control or the Model Predictive Controller which at the end interface to the hardware to generate the motions and forces. For a closed loop operation, the state feedback is needed as input.

▪ Both components interact with a higher-level component or user interface to receive the commands which trigger recording, learning, and execution with new start and goal poses of the DMPs.

- **Image Acquisition**:

○ *General Description*: this component is responsible for capturing and processing image data from supported sensors, such as Photoneo[5] and ZED cameras[6]. It ensures reliable image acquisition through custom ROS2 interfaces defined within the component itself. The system enables both synchronous and asynchronous data capture, making it suitable for a variety of perception tasks, such as segmentation and other image-based analyses.

○ *IN/OUT*:

▪ Inputs to the Image Acquisition Component:

- *Camera Sensor Data*: image data captured from supported cameras (e.g., Photoneo and ZED).

▪ Outputs from the Image Acquisition Component:

- *Captured Image Data*: processed image data ready for further perception tasks (e.g., segmentation)

○ *Interaction with other ROS2 Nodes*:

▪ The Image Acquisition component interacts with perception nodes (e.g., segmentation nodes) by providing captured image data. It interfaces with external camera drivers (e.g., Photoneo and ZED) to facilitate data capture and processing. It also communicates with other nodes in the system to

[5] https://www.photoneo.com/

[6] https://www.stereolabs.com/en-es/products/zed-2

Funded by the European Union

support synchronous or asynchronous data handling for efficient task execution.

- **Segmentation**:
  - *General Description*: this component processes an **RGB image** of the scene using a **deep learning-based algorithm**. Its primary function is to identify and separate objects within the scene by generating a list of masks. These masks enable further processing steps such as object recognition, tracking, or manipulation, making it a crucial part of the perception pipeline.
  - *IN/OUT*:
    - Inputs to the Segmentation Component:
      - RGB Image: an image of the scene, typically captured by a camera such as Photoneo or ZED with the help of the "Image Acquisition".
    - Outputs from the Segmentation Component:
      - List of Masks: a list of masks identifying and separating different objects within the scene for further processing.
  - *Interaction with other ROS2 Nodes***:**
    - The Segmentation component interacts with **Image Acquisition** Nodes by receiving RGB images for processing.
    - The output masks are passed to other perception or manipulation nodes that use them to identify objects and perform further tasks like Object Recognition or Interaction. In addition, serve as an input to the "Pose Estimation".
- **Pose Estimation**:
  - *General Description*: this component estimates the pose (position and orientation) of segmented objects in a scene, relative to the camera. It uses the **CAD** models of the objects in the scene to perform this estimation. The component provides a set of configurable parameters, which can be adjusted to fine-tune the performance of the pose estimation algorithm for different use cases. The system takes as input a list of segmentation masks (from the **Segmentation** component) and sensor data, such as RGB images and coordinate maps (from the **Image Acquisition** component). The output is a list of **6D poses** (position and orientation) for each segmented object, which is then passed to the **Grasping Point Estimation** component for further processing.
  - *IN/OUT*:
    - Inputs to the Pose Estimation Component:
      - *Segmentation Masks*: masks identifying the segmented objects in the scene, obtained from the **Segmentation** component.
      - *RGB Image*: the image captured by the camera, provided by the **Image Acquisition** component.
      - *Coordinate Maps*: additional sensor data that helps in estimating the objects' poses with respect to the camera. This is provided by the **Image Acquisition** component.
    - Outputs from the Pose Estimation Component:

Funded by the European Union

- *6D Poses*: a list of poses (position and orientation) for each segmented object, which is used by the **Grasping Point Estimation** component.

  - *Interaction with other ROS2 Nodes***:**
    - The Pose Estimation component interacts with the Segmentation component to receive segmentation masks that define the objects to be processed.
    - It also communicates with the Image Acquisition component to receive RGB images and coordinate maps for pose estimation.
    - The output of this component, the 6D poses, is passed to the "Grasping Point Estimation" component for the calculation of grasping points.

- **Grasping Point Estimation**:
  - *General Description*: this component calculates potential grasping points for objects in a scene. It takes as input the scene's point cloud, segmentation masks (from the segmentation component), and the corresponding CAD model or primitive of each object. The component begins by computing the objects' poses relative to the camera, then identifies potential grasping points for each object. The result is a set of candidates grasping points, which is passed to the grasping point selection component for further processing and decision-making.
  - *IN/OUT*:
    - Inputs to the Grasping Point Estimation Component:
      - *Point Cloud*: a 3D point cloud of the scene, typically provided by a depth camera or LiDAR sensor.
      - *Segmentation Masks*: masks identifying the segmented objects in the scene, obtained from the **Segmentation** component.
      - *CAD Model or Primitive*: the CAD model or geometric primitive representation of the object to aid in grasping point calculations.
    - Outputs from the Grasping Point Estimation Component:
      - Set of Candidate Grasping Points: A list of potential grasping points for each object in the scene.
  - *Interaction with other ROS2 Nodes***:**
    - The **Grasping Point Estimation** component interacts with the **Segmentation** component to obtain segmentation masks for each object in the scene.
    - It also relies on the Point Cloud data to calculate object poses and determine where the objects are in 3D space.
    - The output, candidate grasping points, is then passed to the **Grasping Point Selection** component for further refinement.
- **Grasping Point Selection**:
  - *General Description*: this component is responsible for selecting the optimal grasping point from a list of candidate points, which is crucial for the robot to pick up an object effectively. It takes as input a list of candidates grasping points (generated by the **Grasping Point Estimation** component) and the scene's point cloud. By calculating

Funded by the European Union

relevant features from the input data, it determines and returns the best grasping point to be used for object manipulation.

- *IN/OUT*:
  - Inputs to the Grasping Point Selection Component:
    - *Candidate Grasping Points*: a list of potential grasping points for each object, provided by the Grasping Point Estimation component.
    - *Point Cloud*: a 3D point cloud of the scene, used to assess the environment and refine grasping point selection.
  - Outputs from the Grasping Point Selection Component:
    - Outputs from the Grasping Point Selection Component:
      - Optimal Grasping Point: the selected grasping point for the robot to execute, chosen based on the calculated features.
- *Interaction with other ROS2 Nodes*:
  - The Grasping Point Selection component receives candidate grasping points from the Grasping Point Estimation component.
  - It also interacts with the Point Cloud data to evaluate the feasibility of grasping points and select the optimal one.
  - The selected grasping point is then passed to the Grasping Execution component (or similar control components) to guide the robot in grasping and manipulating the object.

- **Contact Classifier**:
  - *General Description*: this component implements a novel approach for continuous and incremental learning of patterns in proprioceptive sensory data, such as manipulator joint torques or force sensor readings. Its primary function is to discriminate different contact situations during assembly tasks and raise an alert if these situations do not match previously learned patterns. The component monitors the sensor signal related to the applied forces and torques, which must have a sufficiently high sampling frequency (ideally ≥ 1kHz). Based on this, it outputs a Boolean signal indicating whether the input matches a previously seen pattern, along with the **assigned integer ID** of the matched category. The component connects to the hardware interface for the sensors, providing real-time sensory data. The output Boolean and category ID are then used by other components, like the **Adaptive MPC**, to adjust control behaviour based on the current uncertainty in the assembly process.
  - *IN/OUT*:
    - Inputs to the Contact Classifier Component:
      - *Proprioceptive Sensor Signal*: data related to the applied forces and torques, with a high sampling frequency (≥ 1kHz).
    - Outputs from the Contact Classifier Component:
      - *Boolean Signal*: a signal indicating whether the sensor data matches a learned pattern (True if matched, False if not).
      - Assigned Integer ID: the ID of the matched category, representing the identified contact situation.

Funded by the European Union

- *Interaction with other ROS2 Nodes*:
  - The Contact Classifier connects to a hardware interface to receive proprioceptive sensory data (joint torques, force sensors, etc.).
  - The Boolean signal and integer ID outputs are sent to other components like the **Adaptive MPC**, allowing the controller to adapt its behaviour based on the current uncertainty in the assembly task.

- **Bin/Plane Location**:
  - *General Description*: this component is responsible for identifying the container (bin) or plane on which the objects to be manipulated are located. It processes point cloud data to accurately detect the container or plane and outputs its **6D pose** (position and orientation). This precise localization is crucial for enabling efficient and accurate manipulation of objects during tasks such as picking and placing.
  - *IN/OUT*:
    - Inputs to the Bin/Plane Location Component:
      - *Point Cloud Data*: a 3D point cloud of the scene, typically captured by a depth camera or LiDAR sensor
    - Outputs from the Bin/Plane Location Component:
      - *6D Pose of Container/Plane*: The position and orientation of the identified container or plane, allowing for precise localization in the scene.
  - *Interaction with other ROS2 Nodes*:
    - The Bin/Plane Location component processes the point cloud data (from "Image Acquisition") to detect the container or plane and calculate its pose.
    - The output, the 6D pose, can be used by other components involved in manipulation tasks, such as the **Grasping Point Estimation** and **Grasping Point Selection** components, for better task execution based on the precise location of the objects.

- **Barcode Reader**:
  - *General Description*: this component interfaces with a **Zebra barcode scanner** to read barcodes from objects in the environment. It processes the scanned data and outputs the barcode as a string, enabling identification and tracking of objects in the system. This component is essential for systems requiring object recognition and traceability based on barcode data.
  - *IN/OUT*:
    - Inputs to the BarCode Reader Component:
      - *Barcode Scanner Data*: data from the Zebra barcode scanner.
    - Outputs from the BarCode Reader Component:
      - *Scanned Barcode String*: a string representing the barcode data, used for object identification and tracking.
  - *Interaction with other ROS2 Nodes*:
    - The BarCode Reader communicates with the Zebra barcode scanner to read barcode data.

Funded by the European Union

        - The barcode string output can be passed to other components, enabling object identification and integration into broader system processes, such as tracking or inventory management.
- **Object Size Classification**:
    - *General Description*: this component is responsible for measuring the object of interest and classifying it based on its size. The classification can be performed by evaluating the overall size, the largest dimension, or by using a threshold on a specific dimension (e.g., length or diameter). This component is crucial for determining the appropriate handling and manipulation strategies for different objects based on their physical characteristics.
    - *IN/OUT*:
        - Inputs to the Object Size Classification Component:
            - *Object Measurements*: data representing the measurements of the object (e.g., dimensions like length, width, height, or diameter).
        - Outputs from the Object Size Classification Component:
            - *Size Classification*: a classification indicating the object's size category based on the selected criteria (overall size, largest dimension, or specific dimension threshold).
    - *Interaction with other ROS2 Nodes*:
        - The Object Size Classification component processes object measurements to classify the object. The data is acquired from the "Image Acquisition"
        - The size classification output can be used by other components, such as the Manipulation components, to tailor the handling/placing strategy based on the object's size.
- **Soft Gripper Control**:
    - *General Description*: this component manages the operation of a self-contained gripper. It communicates with the gripper via an **M8 cable**, which provides both the power input and transmits the necessary data signals for its operation. The component enables control over the gripper by setting the operating pad voltage, toggling the voltage on/off, and receiving status messages, including potential breakdown information. The M8 cable connects to a control box, which is equipped with a jack connector for power supply and a USB port for controlling the gripper via serial communication. This control box interfaces with a ROS2-enabled PC for full integration into the robotic system.
    - *IN/OUT*:
        - Inputs to the Soft Gripper Control Component:
            - *Power Input (M8 Cable)*: provides power to the gripper.
            - *Data Signals (M8 Cable)*: transmits necessary data for controlling the gripper (operating pad voltage, on/off toggles, etc.)
        - Outputs from the Soft Gripper Control Component:
            - *Operating Pad Voltage*: sets the voltage applied to the gripper's operating pad.

Funded by the European Union

- *Voltage Toggle*: Allows toggling the voltage on or off for controlling the gripper's engagement.
- *Breakdown Information (String Message)*: a message indicating any potential issues or breakdowns in the gripper system.

- *Interaction with other ROS2 Nodes*:
  - The Soft Gripper Control component communicates with the control box through the M8 cable to manage power and control signals.
  - Through ROS topics, the component can adjust the operating voltage, toggle the voltage, and provide breakdown information for troubleshooting.
  - The control box connects to a ROS2-enabled PC, enabling interaction with other system components and controllers.

- **FBG Sensor Monitoring**:
  - *General Description*: this component is responsible for publishing data acquired from the **FBG sensor interrogator** (a data acquisition device for Fiber Bragg Grating sensors). It can provide time series data of wavelength for all the FBG available channels. The component utilizes **RCLPY** as the ROS2 client library to communicate and provide access to the data, allowing for easy integration into the robotic system.
  - *IN/OUT*:
    - Inputs to the FBG Sensor Monitoring Component:
      - *FBG Sensor Data (from Interrogator)*: data from the interrogator, which includes wavelength information from the FBG channels or the full spectrum.
    - Outputs from the FBG Sensor Monitoring Component:
      - *Wavelength Data*: Time series wavelength data from the available FBG channels.
  - *Interaction with other ROS2 Nodes*:
    - The **FBG Sensor Monitoring** component interfaces with the FBG sensor interrogator to acquire the data.
    - The processed wavelength and spectrum data are published and can be accessed by other components or ROS nodes that require this information for further analysis or control. Mainly in “Grasp Point Selection” and “Grasp Point Estimation”.
    - This component uses ROSYPY as the ROS2 client library to publish data and interact with the ROS2 ecosystem.

The HARTU system allows flexible instantiation of software components, Behaviour Trees, and computational models to tailor solutions for specific robotic applications. Components such as Segmentation, Grasping Point Estimation, or Pose Estimation are instantiated as independent ROS2 nodes, which can be configured at runtime using parameter files. Behaviour Trees are dynamically loaded and executed, enabling modular task planning and execution. For perception and manipulation tasks, models like CAD representations, deep learning-based segmentation models, or DMPs are instantiated with relevant inputs (e.g., point clouds, masks, or training data). This modular approach ensures seamless integration, reusability, and adaptation of components across different robotic platforms and use cases.

Funded by the European Union

11 Software Configuration Management. Update

11.1 Overview

The following sections describe a further round of activities carried out on the *Software Configuration Management (SCM)* System – introduced in D2.1 and based on the GitLab self-managed Core version (open-source Community Edition) publicly reachable at the endpoint https://gitlabpa.eng.it - to support the integration and validation of the solutions in the *real-world* scenarios.

11.2 Use of Docker

In HARTU, ROS 2 nodes and auxiliary components (hereinafter both called *components*) have been encapsulated within Docker images. The choice fell on Docker after considering its widely recognized portability, efficiency and security properties, thanks to which it allows for reproducible deployment on heterogeneous target environments greatly simplifying dependencies and configuration management.

The dockerization activities began with the analysis of the software dependencies for the *components*. The aim was to identify those libraries used by more than one component and rank them based on their usage frequency. With this information, an optimised tree of reusable base Docker images was defined. In it, the larger the number of components using a certain dependency, the closer the dependency is to the root node. For simplicity, the following diagram only shows an excerpt of it.

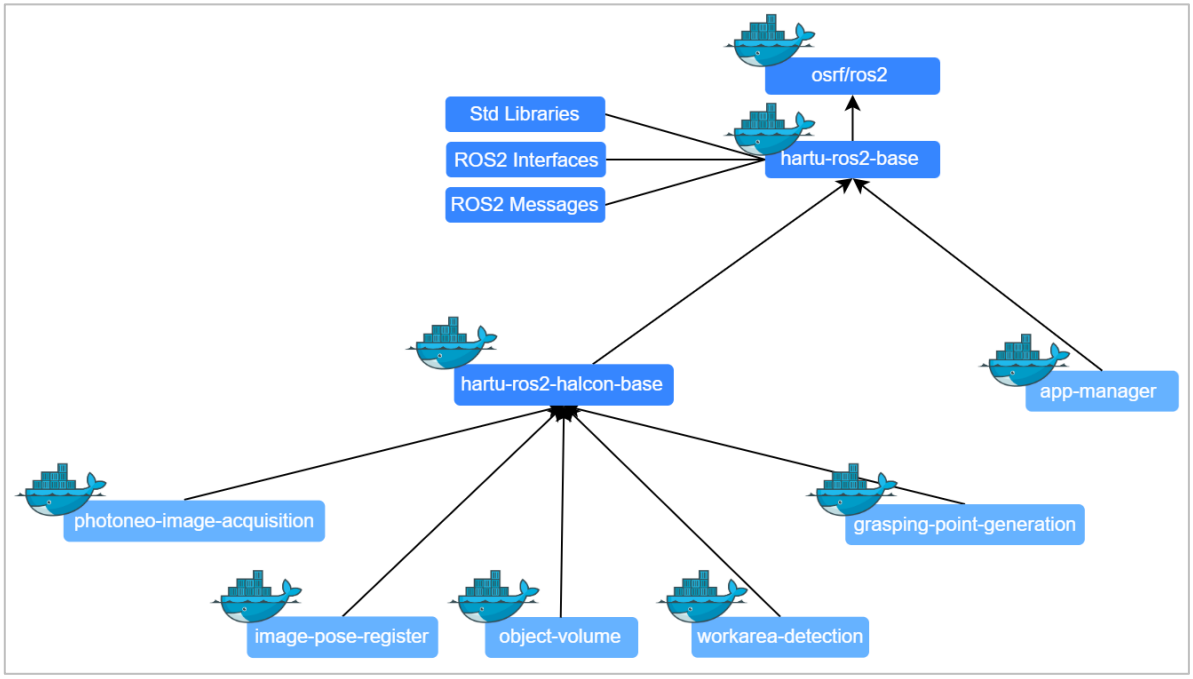


Figure 84 - Tree of Docker images showing the three main base images and some of the child ones

Funded by the European Union

The root node is the official `osrf/ros2` docker image, which provides the basic ROS 2 environment on the specified operating system. This image is used as direct base for the `hartu-ros2-base` docker image, which adds a set of HARTU-specific artifacts common to the majority of child images (standard libraries, ROS2 interfaces and messages). This image is in turn extended by `hartu-ros2-halcon-base`, which adds the HALCON Computer Vision libraries. Once the base docker images are available, the derived ones – i.e. those corresponding to the ROS 2 Nodes - can be created. For instance, in the picture above, five child images requiring the HALCON library can be seen inheriting from `hartu-ros2-halcon-base`, whereas the `app-manager` is built directly on top of `hartu-ros2-base` as it does not depend on HALCON.

Although Docker images are commonly built for cloud native applications, in HARTU there was a need for packaging desktop applications and software interacting with hardware devices. Examples are the `app-manager` and the `photoneo-image-acquisition`; the former featuring a Qt GUI, the latter interacting with a network-connected Photoneo 3D Scanner.

For dockerized GUI applications running on Linux hosts it is sufficient to pass from the host to the container the `DISPLAY` environment variable set to `<linux host IP address>:<X port>`, and grant permissions for the container to access the host X Server. At that point, when the application (an X client) starts container-side, it renders its UI on the X server indicated by the `DISPLAY` variable. On a Windows host, it is possible to use XPRA or install on the host X Servers such as Cygwin, Xming or use MobaXTerm[7]. The latter in particular was used during the test phase.

As far as the tools requiring interaction with the hardware are concerned, it generally depends on the specific protocols used by the *(software, device)* pair. For instance, the Photoneo 3D Scanner is a network device that can be accessed through a discovery service such as Avahi. To allows the Photoneo's software to access this service from a docker container, it is necessary to map a set of `/dev` device files to the container, so that the application can access the required lower-level resources.

Another type of Docker images requires NVIDIA GPU acceleration. To allow the containerized application to access the GPU, it is necessary to install NVIDIA driver and NVIDIA Container Toolkit on the host and have CUDA libraries in the container. Driver and CUDA versions need to be compatible.

[7] Downloaded from https://mobaxterm.mobatek.net/download.html

Funded by the European Union

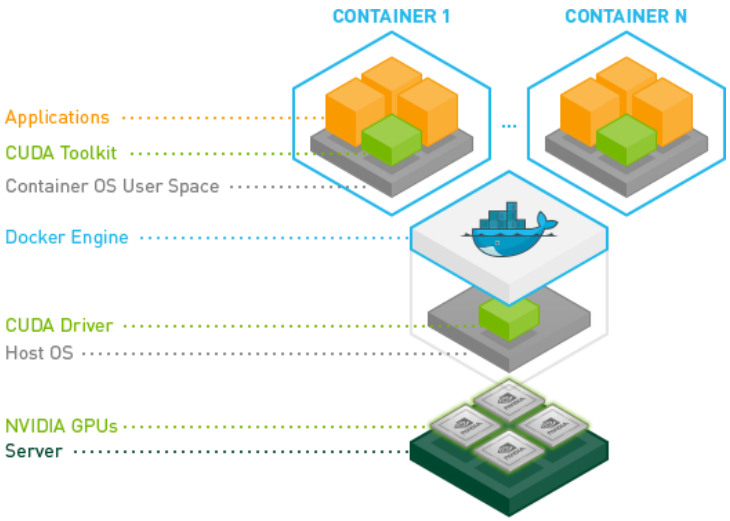


Figure 85 -Nvidia Dependencies [Source: developer.nvidia.com]

At that point, before starting the container, it is necessary to instruct the docker daemon to use the NVIDIA Docker runtime and specify which GPU to use. To this end, Docker CLI arguments are available.

Once ready, the set of Docker images and configuration files specific to a use case can be grouped in a reusable *Docker Compose* file, thus relieving the user from the burden of configuring and launching the single images separately on every occasion. Furthermore, in the context of a docker compose, the dockerized ROS 2 nodes are able to interact with each other within the docker network via the FastDSS middleware. If it were necessary to have them interact also with non-dockerized ROS2 nodes, FastDDS can be configured properly.

```
docker-compose.yml
services:
  app_manager:
    image: reglabpa.eng.it/hartu/apro/mod.fl.uit/hartu-app-manager:0.0.1
    environment: ···
    volumes: ···
    command: ···
    depends_on: ···

  photoneo_image_acquisition:
    image: reglabpa.eng.it/hartu/apro/mod.al.per/photoneo-image-acquisition:0.0.2
    environment: ···
    volumes: ···
    command: "ros2 launch photoneo_image_acquisition photoneo_node.launch.py"

  image_pose_register:
    image: reglabpa.eng.it/hartu/apro/mod.al.per/image-pose-register:0.0.1
    volumes: ···
    command: "ros2 run image_pose_register image_pose_register_node"

  object_volume:
    image: reglabpa.eng.it/hartu/apro/mod.al.per/object-volume:0.0.1
    volumes: ···
    command: "ros2 launch object_volume object_volume_node.launch.py"
```

Figure 86 - Excerpt of docker compose file prepared for one of the use cases

Funded by the European Union

11.3 CI/CD Pipelines

To streamline the use-cases solutions integration process, a system of *parent-child* pipelines has been implemented on GitLab. More precisely, each project contains a *parent* pipeline composed of trigger jobs. Each trigger job launches a *child* pipeline corresponding to a specific component (again, ROS 2 node or auxiliary module). The execution of the child pipelines is controlled by a set of customized invocation rules, that guarantee that only the required child pipelines are launched when needed (e.g., only those for which there are changes to the code). Moreover, parent-child pipelines are particularly well suited for monorepos – as in the case of HARTU - in which different parts of the project need to be built independently and with different requirements. All these properties not only promote modularity but also enable performance increases by allowing child pipelines to run concurrently. The following figure shows three child pipelines (downstream) launched from `stage1` trigger jobs:

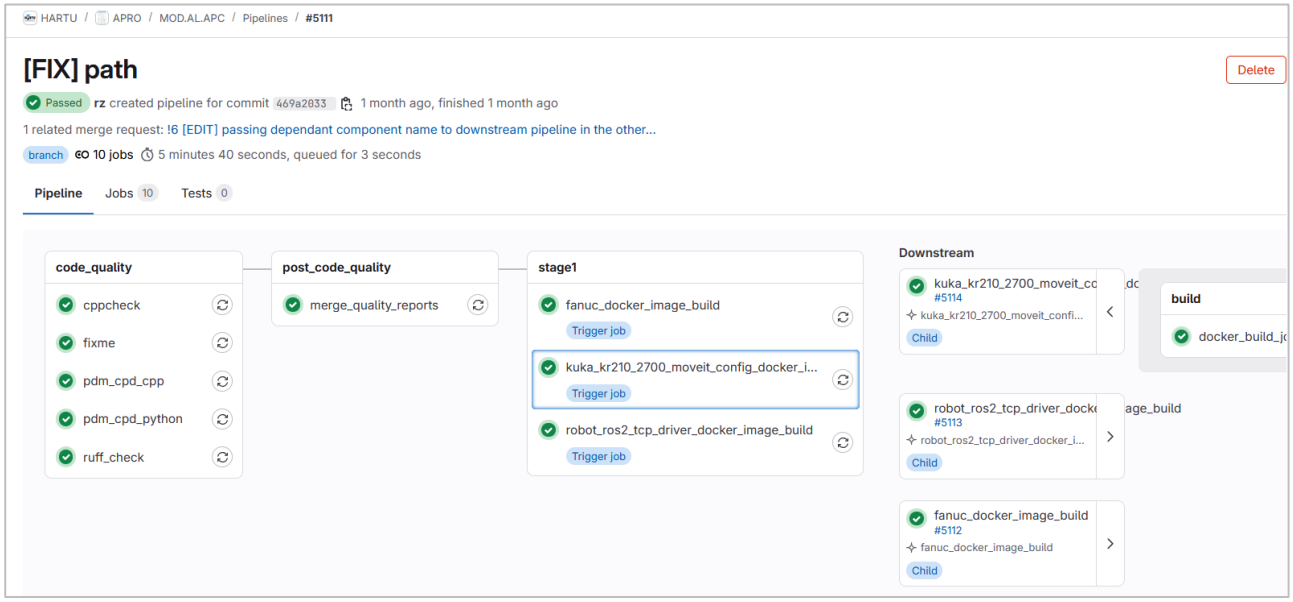


Figure 87 - Example of pipeline with code quality jobs and docker images build

11.3.1 Automated Docker images build with GitLab

Some jobs are dedicated to the building of Docker images. In particular, every time a change is made to the code or to a Dockerfile, these jobs:

1. Download the required dependencies, either from:
    a. The GitLab-integrated Package Registry
    b. An external one, or
    c. Other projects within the same repository
2. Execute the build instructions
3. Tag the built image
4. Push the tagged image to the GitLab-integrated Container Registry

Funded by the European Union

Having Docker images continuously built and pushed to the registry allows end-users and system integrators to quickly deploy software updates on the pilots' target environments. It is in fact often sufficient to update the image tag and restart the corresponding docker service to have the latest version of the software up and running. The following figure shows the GitLab-integrated Container registry with some of the Docker images:

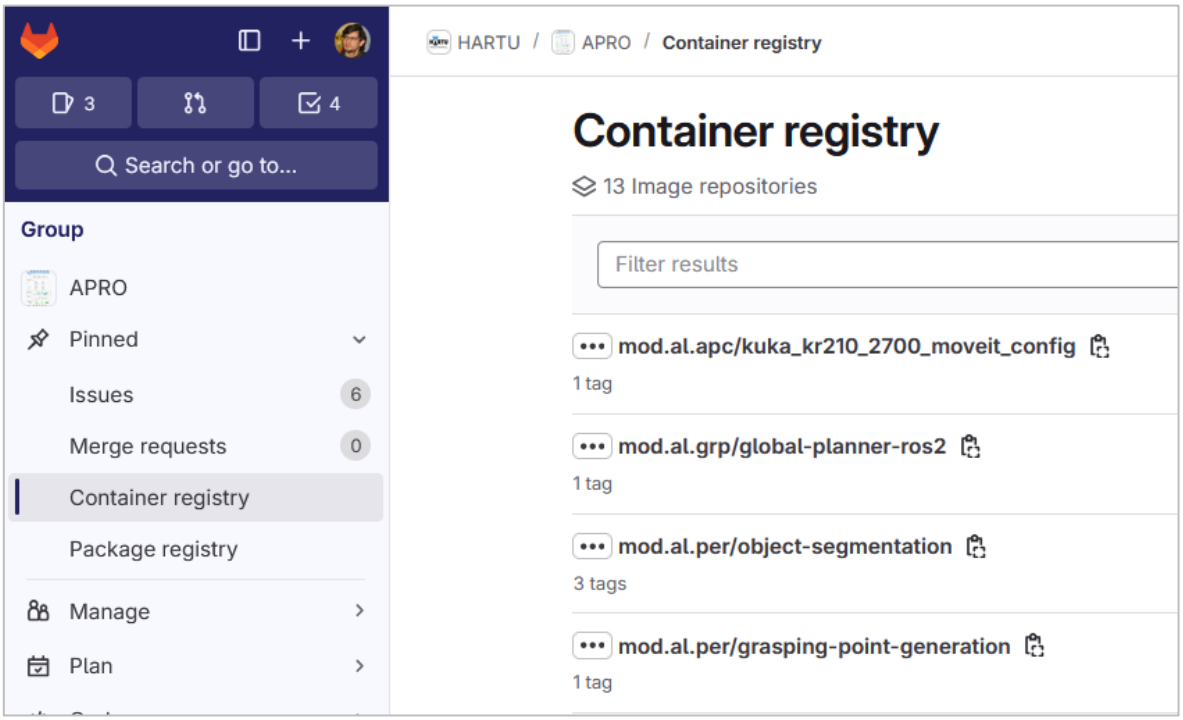


Figure 88 - Docker images are pushed to Gitlab-integrated container registry

11.3.2 Code Quality with GitLab

The aforementioned project pipelines also feature jobs dedicated to **static code quality analysis**. The previous GitLab's Code Climate-based approach – already described in previous deliverables - has now been updated to the latest recommendations. If in the previous version, code quality analysis was reliant on mechanisms and tools that had to be available in Code Climate, now the approach is more open and flexible, allowing DevOps engineers to configure the analysers of their choice. In particular, a number of scanners have been integrated in GitLab as docker images and then used to create a pipeline template to be later imported by all the other pipelines. This templating mechanism allows to keep a central definition of code quality jobs that when updated becomes immediately available to the dependant pipelines.

Funded by the European Union

HARTU / Templates / **Code Quality Pipeline**

develop | code-quality-pipeline / **code-quality-pipeline.yml**

code-quality-pipeline.yml

Edit code-quality-pipeline.yml
pietro.greco authored 2 weeks ago

code-quality-pipeline.yml 3.62 KiB

```yaml
workflow:
  rules:
    # make this pipeline run always, including in merge_request_pipeline
    - when: always

stages:
  - code_quality
  - post_code_quality

.basic_code_quality_job:
  tags:
    - docker
  stage: code_quality
  rules:
    - changes:
      - src/**/*

# CDP: Copy/Duplicate Detector (C++ and Python)
pdm_cpd_cpp:
  extends: .basic_code_quality_job
  image:
    name: registry.gitlab.com/gitlab-ci-utils/gitlab-pmd-cpd:2.5.3
    entrypoint: ['']
  script:
    - export PMDCPD_MIN_TOKENS=100
    - export PMDCPD_LANGUAGE=cpp
    - export PMDCPD_RESULTS=pmd-cpd-results-cpp.xml
    - /gitlab-pmd-cpd/pmd-cpd.sh
```

Figure 89 - Excerpt from Code Quality pipeline template

.gitlab-ci.yml

[FIX] job name and dockerfile for grasp planner
pietro.greco authored 2 months ago

.gitlab-ci.yml 882 B

```yaml
include:
  - project: 'hartu/templates/code-quality-pipeline'
    ref: master
    file: 'code-quality-pipeline.yml'

stages:
  - code_quality
  - post_code_quality
  - stage1
```

Figure 90 - Code Quality pipeline template being imported from another pipeline

Here is a list of available scanners:

- PDM CPD (C++ and Python)
- Ruff (Python)

Funded by the European Union

- CppCheck (C++)
- Fixme

Finally, it is worth to mentioned that compared to the previous one, also the architectural implementation results enhanced, now being lighter and more efficient.

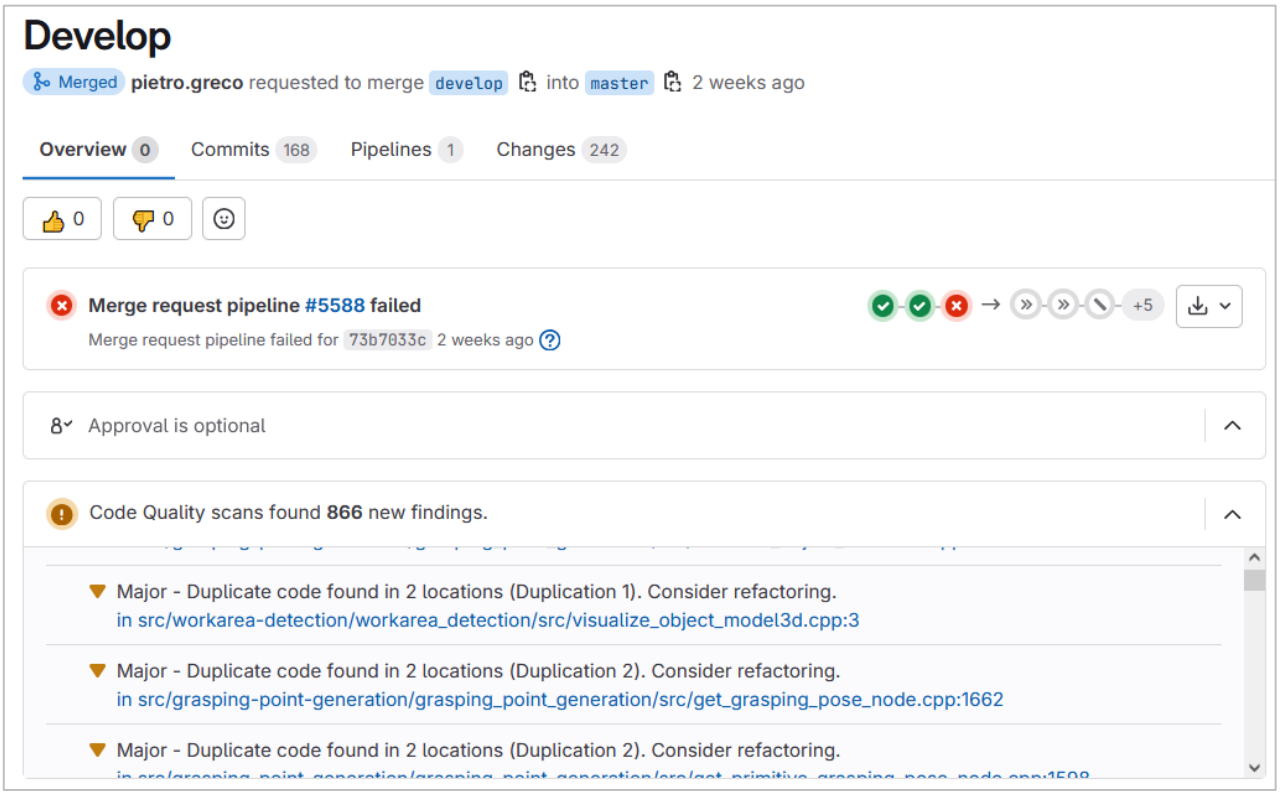


Figure 91 - Code Quality Issues report in Merge Request widget

Last but not least, a set of scripts capable of generating **statistics about code quality** has been developed. More precisely, they allow for keeping track of code quality improvements over time providing per-module trend insights. To facilitate their work and speed up the integration of clean code, developers have also installed the static code scanning tools used in GitLab on their local workstations, either in the form of IDE plugins or standalone tools.

11.4 Issues Management with GitLab

To ensure structured communication between technical partners and facilitate activities monitoring, the tasks related to technical issues resolution or implementing new features have been managed through GitLab's Issues Management module. From the user interface, members can define new issues entering metadata such as title, description and even design diagrams. It is also possible to set custom labels, start and due dates as well as milestones. At that point, members can have any related discussions through the comments area. Finally, users who have also activated notifications receive emails with updates concerning the Issues (new comments, issue status change and so forth).

Funded by the European Union

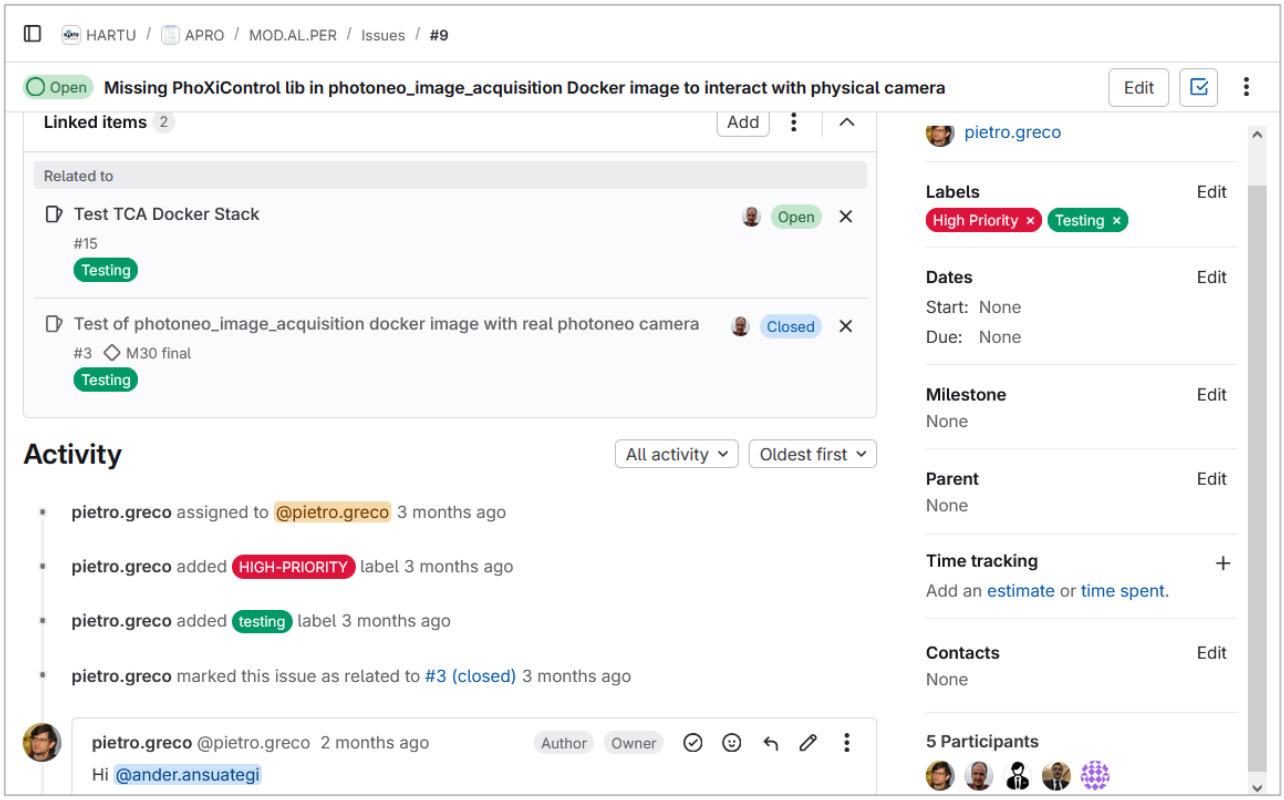


Figure 92 - Example of issue in Gitlab

Funded by the European Union