Project Acronym:	CONVERGING
Grant Agreement number:	101058521 (HORIZON-CL4-2021-TWIN-TRANSITION-01-01 – Innovation action)
Project Start Date	1st September 2022
Project Full Title:	Social-industrial collaborative environments integrating AI, Big Data and Robotics for smart manufacturing.

CONVERGING

Funded by the Horizon 2020 Framework Programme of the European Union

DELIVERABLE

D4.3 – Open integration and communication Architecture - deployment

Dissemination level:	PU
Type of Document	OTHER
Contractual date of delivery	31st Aug 2025
Deliverable Leader	INTRA
WP / Task responsible	WP4, T4.3, T4.4
Keywords:	Software Architecture, Data At Rest, Data In Motion, Digital Twin, System Integration

Executive Summary

This document describes the findings and developments of Work Package 4 - Big Data Pipeline. The deliverable covers the work developed in WP4 from M31-M36, which is August 2025.

During this period, the tasks that were active were Task 4.3 “Differentiable digital twins for optimized Machine Learning and AI based planning” and Task 4.4 “Open integration and communication architecture for reconfigurable production - deployment”. These tasks worked towards achieving the final objectives of WP4 towards the development of the Big Data Pipeline. In particular, the efforts of WP4 were focused on the development of the AI Digital Twin (AIDT) module and the Integration and Communication Architecture led by INTRA.

The AI Digital Twin (AIDT) module has been developed on top of Visual Components 4.0 and 5.0, which accelerated the deployment of the Digital Twin for the pilot cases, including virtual representation of the shop floor, simulation libraries, integration with planning modules, and communication with ML/AI frameworks. During this last period, experimentation with other simulation frameworks was also done due to the limitations of the Visual Components software in real-time robot control and navigation in confined spaces. These involve MoveIt! and NVIDIA ISAAC sim. Additionally, the functionalities required to address the CONVERGING requirements have been finalized. Improvements were achieved in the simulation and virtualization engines, the path planning, and in the AI/ML components.

The final part of the deliverable presents the final version of the CONVERGING Integration and Communication architecture. The CONVERGING architecture is modular and reconfigurable and serves as the backbone for connectivity and interaction among production resources, software, and modules. Integration and Communication Architecture includes the interfaces and information flow between different software modules as well as a set of supporting tools and methodologies that have been used towards the development of the integrated CONVERGING system.

Table of Contents

List of Figures

List of Table

Definitions, Acronyms and Abbreviations

Acronym/Abbreviation	Description
AI	Artificial Intelligence
AIDT	AI Digital Twin
AISC	AI Station Controller
AM	Additive Manufacturing
API	Application Program Interface
ARBA	Autonomous Robot Behaviour Adjustment
CAD	Computer Aid Design
CDC	Change Data Capture
CLI	Command Line Interface
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CRC	Collaborative Robot Control
DAR	Data at Rest
DC	Digital Component
DIM	Data in Motion
DWR	Dynamic Work Reorganization
DS	Digital Shadow
DTI	Digital Twin Instance
DT	Digital Twin
DTP	Digital Twin Prototype
DWR	Dynamic Work Reorganization
EOC	Execution Orchestrator Component
GMC	Graph Manager Component
GPU	Graphic Processing Unit
HGC	Hand Guided Control
HPCM	High Payload Collaborative Manipulator
HRI	Human Robot Interaction
JSON	JavaScript Object Notation
LiDAR	Light Detection and Ranging
MACI	Multi-Actor Contextual Interfaces
MD	Markdown
ML	Machine Learning
MQTT	Message Queuing Telemetry Transport
OLP	Off-Line Programming
OPC-UA	Open Platform Communications Unified Architecture
OPL	Open Pilot Line
PAM	Perception & Autonomy module
PDCA	Plan-Do-Check-Adjust
PFL	Power and Force Limited
PM	Process Modelling

Acronym/Abbreviation	Description
PPO	Proximal Policy Optimization
RCC	Robot Client Component
RGB-D	Red-Green-Blue-Depth
REBA	Rapid Entire Body Assessment
RiTRRT	
RMI	Robot Manager Interface
RNN	Recurrent Neural Network
ROS	Robot Operating System
BiTRRT	Bi-directional Transition-based Rapidly-exploring Random Trees
RRT	Rapidly-Exploring Random Tree
RTO	Research and Technology Organization
RULA	Rapid Upper Limb Assessment
SAM	Safety Assessment and Monitoring
SRP/CS	Safety Related Parts of the Control System
SSM	Speed and Separation Monitoring
TCP	Transmission Control Protocol
TDM	Teaching by Demonstration Module
UI	User Interface
UML	Universal Modelling Language
URDF	Universal Resource Description Format
UXE	User Experience and Ergonomics
VC	Visual Components Platform
VC 4.0	Visual Components Platform v.4.0
VC 5.0	Visual Components Platform v.5.0
VRC	Virtual Robot Controller
VRAM	Video Random Access Memory
WP	Work Package

1 Introduction

1.1 Objectives of the deliverable

The deliverable presents the final outcome of tasks T4.3 “Differentiable digital twins for optimized Machine Learning and AI based planning” and T4.4 " Open integration and communication architecture for reconfigurable production – deployment”. In particular, this deliverable presents the final prototype of the AIDT (AI Digital Twin) software module and the Open integration and communication Architecture – deployment. The focus of the document is the presentation of the latest findings and developments of the two aforementioned tasks, while also remaining self-contained.

1.2 Structure of the deliverable

The deliverable is structured in sections that correspond to each of the two work package tasks reported. Each section begins with a brief module introduction, providing a context for their role in CONVERGING. The section continues with the design, which includes the deployed architecture and technologies, to continue with the functionalities, development status, and next steps.

More specifically, section 2 presents the AI Digital Twin (AIDT) module, led by VIS that has been developed in Task 4.3. Section 3 corresponds to Task 4.4 leaded by INTRA which focuses on the Integration and Communication architecture.

1.3 Relationship with other CONVERGING work

CONVERGING D4.3, titled “CONVERGING Big Data Pipeline and Hybrid Digital Twins Open integration and communication Architecture - deployment,” builds on top of the developments and continues its development by leveraging the groundwork done in D4.1 and guided by the work of WP2 and particularly D2.2 titled “Design of the Reference Software Architecture”. The progress in WP4 has been closely linked to advancements in other technical work packages (WP3, WP5, and WP6) and the integration efforts in the four demonstrators under WP7. This progress will be utilized for the integrated development in Pilot Cases, which takes place in WP7. Figure 1 provides a graphical representation of the key relationships of D4.3 with other CONVERGING work.

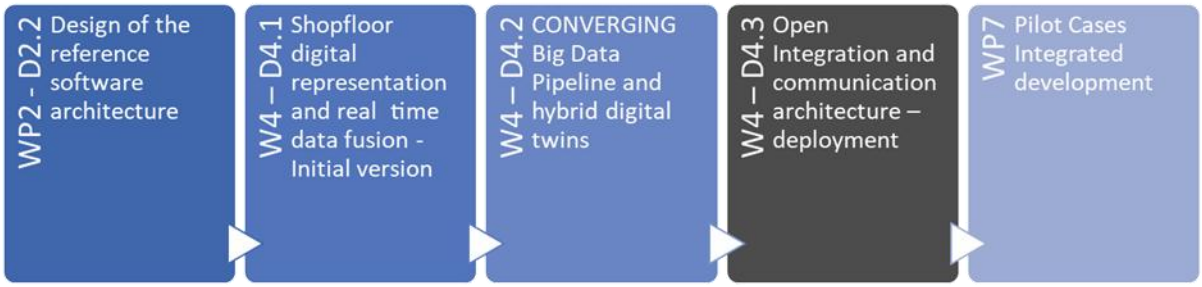

Figure 1: Relationship with other CONVERGING Work

2 AI Digital Twin (AIDT) - Final Version

The development of the AIDT has been done on top of Visual Components 4.0 (VC 4.0). During the development of WP4, incremental releases (r4.8, r4.9, and r4.10) of VC 4.0 have been used in CONVERGING's development tasks, taking advantage of the latest simulation technology to develop and implement CONVERGING's solutions. Furthermore, Visual Components 5.0 (VC 5.0), which provides a new simulation technology extended with new interfaces and visualization solutions, has become available within the consortium, allowing project development and implementation to be transferred to this platform when possible. Despite the advancements in simulation technology, the UI (User Interface) has retained its original structure, facilitating the users' adoption of the new product family. Additionally, due to the limitations of the VC4.0-VC5.0 identified in real-time robot control and planning in confined spaces, additional frameworks were tested by LMS, involving MoveIt! and ISAAC sim. Details will be presented in the following sections.

2.1 Progress (M36)

This section reports the new developments and findings from M31-M36.

2.1.1 AIDT progress

The AIDT has followed an incremental development process that was completed by M36 (August 2025). D4.1 and D4.2 presented the identified challenges, and the development has been aligned with the work in WP 3 and WP 4.

2.1.1.1 Robot motion controller

Updates to the robot controller have brought the simulation environment closer to the real robot system, meeting the CONVERGING requirements. The new motion controller in VC 5.0 brings a comprehensive overhaul of the motion system, delivering significantly faster execution times and enhanced overall performance. It introduces smoother motion interpolation, resulting in more realistic and fluid robot behaviour that closely mirrors real-world dynamics. Users now benefit from greater control over robot configurations and path planning, enabling more precise and adaptable automation workflows. The system also features improved kinematics, particularly in handling complex configurations and edge cases that previously posed challenges. Additionally, it brings robust support for multi-driver motion, coordinated movements, and external axes, making it ideal for advanced robotic applications involving synchronised tooling, gantries, or auxiliary axes.

CONVERGING configurations like the KAWADA two-arm robot (detailed in section 2.2.3) and the Aeronautics Pilot Case remote inspection robot (detailed in section 2.2.4) now have more accurate simulations.

2.1.1.2 Trajectory planner

The solver capabilities have been expanded to meet the needs of CONVERGING's pilots, improved to handle a broad range of trajectory planning functions, including collision detection and resolution, singularity avoidance, handling joint limit violations, and reachability analysis. These features are applicable across process paths, search paths, and intermediate motions (via paths).

To improve collision detection and path planning, colliders have been implemented. Colliders are simplified geometric representations (e.g., boxes, spheres, convex hulls) used to approximate the shape of 3D models, drastically reducing computational complexity compared

to using full resolution meshes, and enabling faster and more efficient simulation and solver performance.

The solver system recalculates the robot joint configurations, auxiliary axes, and approach/exit points. Although the solver capabilities can be accessed through the VC UI interface, as presented in detail in the section 2.2.4.3, interfaces have been developed to integrate the solver system into the CONVERGING framework.

These interfaces, deployed through VC API, allow several actions which can be accessible through ROS2 and MQTT, including, robot program, sequence, and selected statements:

- Open Panel

  Opens the solve panel at Visual Components UI, allowing the user to manually trigger solving operations for selected paths, sequences, or the entire program.

- Run Solver Action

  Executes the solver that calculates optimal robot joint configurations for via points (intermediate points between main process points), ensuring collision/free and reachable robot poses for all via points in the program.

- Solve Program Process Paths

  Adjust robot configurations and auxiliary axes of all process paths to ensure reachability and avoid collisions, running the solver for all process paths in the entire program.

- Solve Program Search Paths

  Runs the solver for all search paths in the program, ensuring search motions are feasible and safe. Search paths are used for touch sensing or seam finding.

- Solve Program Via Paths

  Runs the solver for all via paths (intermediate moves between process or search paths) in the program, optimizing, optimizing robot joint positions for smooth transitions and collision avoidance.

- Solve Selected Process Paths

  Runs the solver only for the selected process paths. This process is useful when it is necessary to modify a single path and update only that path, rather than the entire program.

- Solve Selected Search Paths

  Runs the solver only for the selected search paths, updating feasibility for specific search motions without recalculating everything.

- Solve Selected Via Paths

  Runs the solver only for the selected via paths, optimizing intermediate moves for selected transitions.

- Solve Sequence Process Paths

Runs the solver for all process paths within the selected sequence (subroutine), which is useful when working on a specific section of the program.

- Solve Sequence Search Paths

  Runs the solver for all search paths within the selected sequence, ensuring all searches in that sequence are valid.

- Solve Sequence Via Paths

  Runs the solver for all via paths within the selected sequence, optimizing intermediate moves for that sequence only.

2.1.1.3 Python 3 support

Identified in D4.2 and to facilitate the integration with other CONVERGING modules that exploit Python 3 technologies, it has become available in VC 5.0, with improved syntax and support for external libraries such as VRC (Virtual Robot Controller), ML/AI to enable big data integrations tasks. Interfaces developed within CONVERGING have been ported to the API to enable access through Python 3.

2.1.1.4 Ergonomics calculation

Continuing the work on calculating REBA and RULA within the VC platform, the human models have been improved to provide more accurate simulations and ergonomic assessments using developed algorithms for REBA and RULA. The integration of runtime interfaces via MQTT enables real-time data retrieval from operators for these calculations.

Further research on the developed capabilities within CONVERGING connecting to cloud-based dedicated ergonomic tools, like WorksErgo[1], enhanced the big data aspect of ergonomics evaluation, complementing the previous work developed in the project, and extending the previous work, and getting more accurate ergonomics calculations.

2.1.2 Trajectory navigation for complex environments

2.1.2.1 Problem presentation

As mentioned before, due to the limitations of VC4.0/VC5.0 to provide fast (real-time) collision free robot path planning, on top of the VIS work, LMS experimented with additional frameworks and planners to identify if this challenge could be overcome. This came more apparent while delivering the first version of the OPLs, and specifically after the IAI one, which involves the path planning of a 10-axis robot, inside a highly restricted area and utilizing real-time generated point cloud data. Despite the new robot planners developed by VIS, the single threaded nature of the VC software limits its speed and effectiveness. The multithreaded approach developed by LMS shows promising results.

2.1.2.2 Solution outline

Our solution for addressing those issues is presented bellow. Larger trajectories can be divided in sub-trajectories, making it easier for path-planners to find a solution in the required time. Those goals can guide the robot using still the classic planners to find more easily a solution without caring about the distance from start to finish. Many of those points also can be shared for leading the robot in different areas. The goals are taking the role of algorithmic

[1] https://www.worksergo.com/

guidance while they still keep the trajectory safe for collisions as its calculated in real time with the octomap data in mind. The following assumptions are made:

1. All the neighbour point to point transitions are easily solvable with classic planners
2. Not all in between goals can have plan between them
3. For each of the points starting from a common start state is always a path trajectory to each of them

Given the non-directional graph structure just described, a structure of connected non directional graph with nodes and edges that represent the goal targets and transitions between them with the appropriate movegroup. Assuming the convention that some points are labelled as free to move, meaning that an appointed target needs to have its the closest point in the graph labelled free. This is because some points in many cases they give very limited joint freedom to the robot to move to, so by custom labelling poses that are easy for the robot to navigate in the space the probabilities of successful plan calculations increase.

The navigation goals can be generated in the preprocess stage by hand or automated (currently the automated version is on development). The process for the graph generation is as follows:

1. Setting up a visualization tool like RVIZ with MoveIt! with octomap of the real robot and environment
2. Set a variety of goals and execute the planning. If the plan executes then save the goal to the graph with additional info about the moved movegroup or preferred speed
3. Export the graph to json file for later use.

During the trajectory calculation, when the target is close to one of the predefined points, the robot uses the appropriate method to fetch its current position from the relevant ROS topics. Based on the end-effector position, it determines its location on the graph and does the same for the target navigation goal. Then, it runs a BFS algorithm to find the shortest path between the current and target nodes, creating the overall execution plan accordingly:

1. Trajectory to the closest free node from the current robot state
2. Generation of trajectories for each of the points in the graph solution path
3. Trajectory from the closest free navigation node in the graph to the actual goal

In total the graph works as a base of knowledge for the robot easily adaptable to new settings and scalable to handle a very big set of nodes. The graph can always change in real time between plans to be enhanced with the new current goals for example. This enables us to do in future real time graph generation within parallel simulation to auto generate the goals while the robot moves this primarily requires appropriate computational power that currently is not available. Each of the edges contain optional additional information about the transition customized movement.

2.1.2.3 Implementation

The system is separated in components as shown in Figure 1 with each handling its separated logic for scalable maintainable structure. The first aspect of the system is optionally providing the JSON file with the appropriate precalculated data. It is optional because the graph without it will be empty starting from scratch will auto generate the JSON on the process. The Graph Manager Component (GMC) is handling the graph load export operations, the graph navigation and implementing functions for easily get access to info about poses. The next component is the Execution Orchestrator Interface (EOI) that is optionally also a ROS node for remote execution of commands by events, handling the coordination between Graph and

other components and providing an easy way for the user to use the system execute commands and create update or save the graph to JSON. Next, there is the Robot Manager Interface (RMI). This is an abstract interface for providing a universal way to communicate with lots of different robot clients for basic operations. Finaly, there is the Robot Client Component (RCC) that implements the GMC interfaces with its own low-level implementation, fields and functionality. This is required because different resources have different interfaces. This final component makes the MoveIt! commands for triggering trajectory calculations and executions. The aforementioned system connects with AISC so commands for navigation goals/poses are feasible in multiple areas inside the fuel tank.

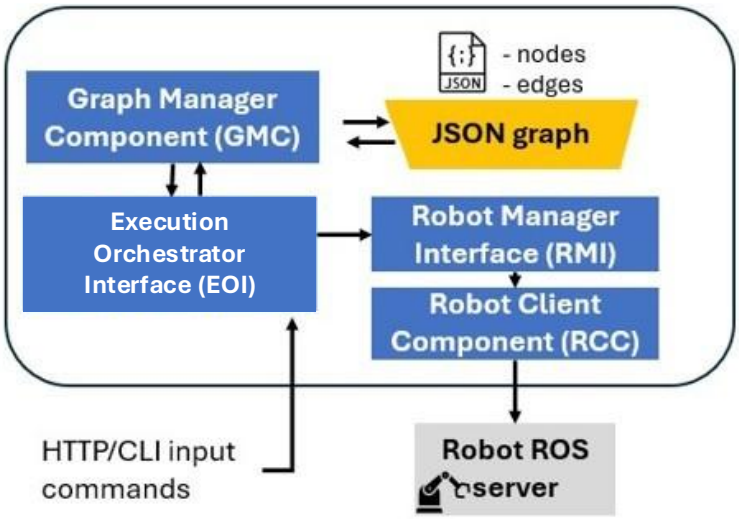

Figure 1: Navigation system structure and subcomponents.

2.1.2.3.1 Graph generation process

As described above, generating the graph first requires setting up the environment with all collision objects and correctly placing the robot using its URDF model. After starting the system and initializing the Robot Client component, which implements the Manager, nodes can be added or removed, and connections between them can be defined using the Command Line Interface (CLI). The backend process of adding a point follows specific steps. First, the Robot Client component is initialized and subscribes to the ROS topic /joints, which continuously listens to the robot's real position in the simulation at a fixed rate. This state is stored internally for fast access. By implementing the appropriate MoveGroups in the MoveIt configuration, the class can directly access the end-effector positions for each MoveGroup independently, allowing separation of movements for more stable trajectories. In the CLI, pressing the 'a' key starts the onboarding process for configuring a new point. During this process, the system gathers information such as the MoveGroup name used to reach this position from the parent node, an optional label for the point, whether it is a free point for movement, and it establishes a connection in the graph with the parent node. The 'p' command is used to set the parent node, typically once or whenever the user needs to redefine the parent node to align with the robot's current position. After adding a node, the parent node is automatically updated to the newly created one for a smoother user experience.

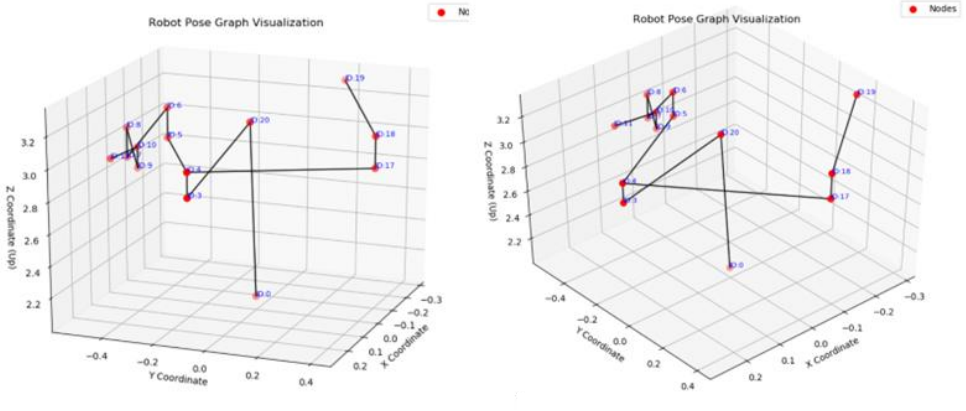

Figure 2: Final data graph for enabling robotic navigation within our desired limits of the tank.

For looking the current state of the graph to the CLI the command is 'l'. And there are other useful commands that I won't analyse in detail but for quick overview:

1. 'a' : adding a node to the graph
2. 'p' : setting parent node to closest for the current robot state
3. 's' : finding and printing the shortest path between 2 nodes that the user enters
4. 'r' : reload the graph from file
5. 'exit' : exiting the CLI and saves the graph
6. 'e' : execution real navigation to a target node
7. 't' : execution real navigation to a target pose in space
8. 'd' : deleting an edge between nodes
9. 'x' : deleting a node with all edges from and to it
10. 'start' : prompts the user to give start and end joints states and movegroup name to return if can be found a trajectory between them.

2.1.2.4 Data types & formatting

All data structures are implemented using Python classes that inherit from the Pydantic library, which provides a convenient way to serialize and deserialize class attributes to and from JSON.

1. **Point** – Represents the 3D position of a point with floating-point fields x, y, and z.
2. **Quaternion** – Represents the rotation with floating-point attributes x, y, z, and w.
3. **MoveGroupState** – Stores the state of a MoveGroup, including:
   - position: a Point object,
   - rot: a Quaternion object,
   - joints: a tuple of floating-point numbers representing the MoveGroup joint states.
4. **Pose** – Represents the complete robot pose and contains:
   - id: an integer identifier,
   - all_ee_poses: a dictionary with the MoveGroup name as the key and MoveGroupState as the value for each available MoveGroup,
   - is_free_to_move_from: a Boolean flag indicating if the point can be used as a free movement origin,
   - is_joint_goal: a Boolean flag defining whether the goal refers to specific joint values or a position/orientation target,

- label: a string used for easier node identification in both the CLI and JSON representation.

5. **Edge** – Defines a connection between two nodes with:
   - source_id and target_id identifying the connected nodes,
   - move_group_for_transition indicating the MoveGroup used for the transition,
   - options: an optional field containing an EdgeOptions instance.
6. **EdgeOptions** – Currently holds additional parameters used for specific movements, such as signal triggering or speed control, allowing custom behavior beyond the default execution logic.
7. **RobotState** – Contains a dictionary where each key is a MoveGroup name, and each value is a list of floating-point joint state values.

For all the above classes, equality and hashing functions are overloaded to enable comparison and detection of duplicate entries based on their field values. Finally, all stored data are serialized into JSON format, forming the complete graph structure that consists of two main elements:

- nodes: a list of serialized Pose objects, and
- edges: a list of serialized Edge objects.

Together, these elements contain all the necessary information to execute any movement between a start state and a target goal.

2.1.2.5 Integration with ROS

The ROS integration is implemented within the manager classes, namely the SimulationRobotManager and RealRobotManager. These classes include ROS listeners that retrieve the real-time joint states in the correct format and store them in a managed RobotState object. They also handle the creation of ROS message commands for goal execution and motion planning through the following functions:

- get_ee_pose_for_group – retrieves the end-effector pose for a specific MoveGroup,
- execute_joint_goal – executes a motion toward a specific joint configuration,
- execute_pose_goal – executes a motion toward a specific position and orientation goal.

Communication with the custom ROS servers, which are connected to the robot and the RViz–MoveIt! simulation setup, is achieved using the rospy and actionlib Python libraries.

The module was successfully integrated with the RIR robot for the IAI use case, performing flawlessly in both trajectory calculation and execution. It effectively addressed the main challenge of this case, which was trajectory generation. The system was tested in multiple confined areas within the tank environment, where inner manoeuvres were required, and it operated as expected during user demonstrations. This use case presented several environmental challenges that the module was specifically designed to handle, such as navigating through a narrow tank entrance and dynamically planning paths within the tank. An additional constraint involved coordinating separate MoveGroups (“default” and “liftkit”), but this was efficiently managed by the proposed approach. Speed options defined at the edges of the graph were used to slow down critical or risky movements and accelerate safer ones, improving both safety and efficiency. Multiple MoveGroups were employed to overcome hardware limitations that prevent real-time position and velocity control of the liftkit column. The MoveGroups used in this setup were “default” (elfin + COMAU Racer), “liftkit”, “racer”, and “elfin”. Figure 9 illustrates the different MoveGroups highlighted during execution.

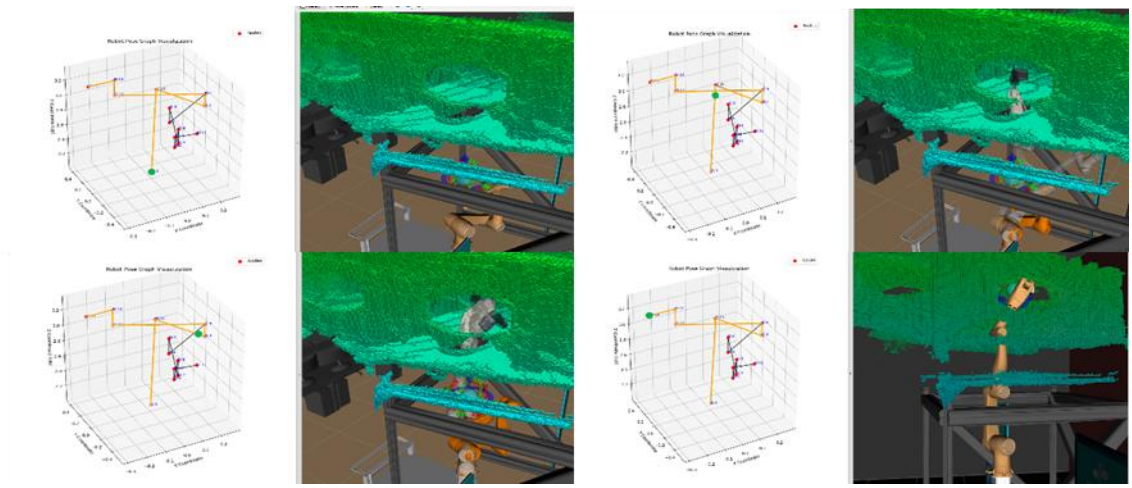

Figure 3: Demo of dynamic execution in RVIZ with Octomap data of the real wing.

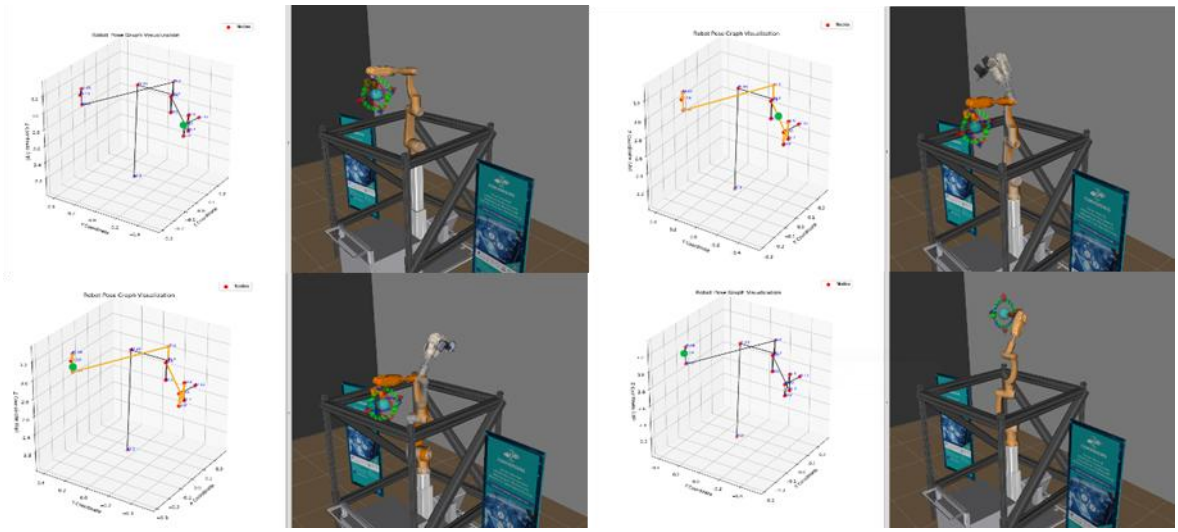

Figure 4: Transition demonstration between graph points.

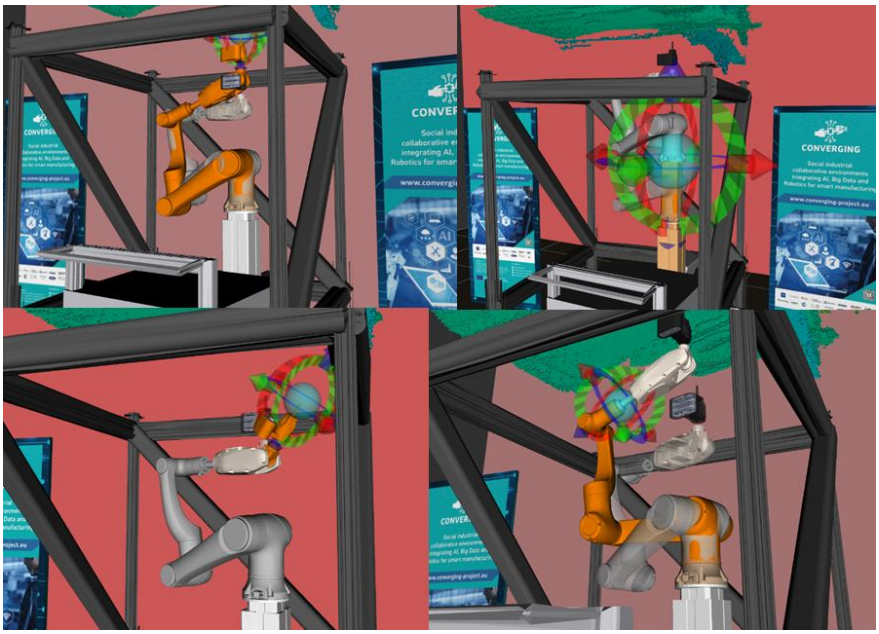

Figure 5: Visualization of different movegroups used for planning.

2.1.2.6 Results

Regarding the tests of the final system on the RIR robot, the completion success rate was approximately 80%. The remaining failures were caused by hardware issues unrelated to the graph system implementation or the planner's trajectory calculations, which in some cases resulted in invalid execution plans.

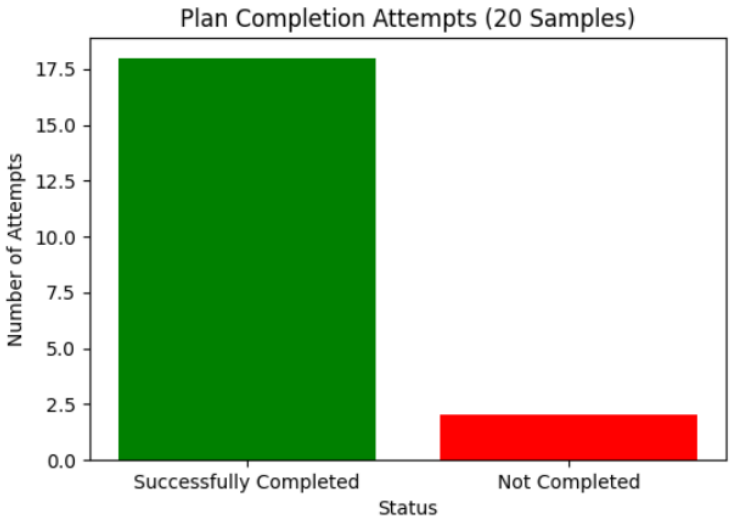

Figure 6: Measured success rate from 20 sample runs to random points in the tank.

2.1.3 Updates on use of AI/ML

Even if the approach described above gave promising results, to scale it up a Reinforcement Learning based approach was also developed by LMS.

2.1.3.1 Simulation software

NVIDIA's Isaac platform (Isaac Sim and Isaac Lab) was used for the development and validation of the reinforcement learning models. Built on the NVIDIA Omniverse ecosystem, Isaac Sim provides a high-fidelity, GPU-accelerated environment for robotics simulation and learning. It leverages the PhysX engine for real-time physics and RTX ray tracing for realistic rendering of camera, LiDAR, and contact sensors. The platform allows direct import of 3D robot models from URDF files using built-in importers, and supports the implementation of robot control logic in Python or C++, seamlessly integrated with ROS 2.

The Isaac Lab framework, built on top of Isaac Sim, provides an open-source, gymnasium-compatible API for reinforcement, imitation, and demonstration learning workflows. It enables large-scale training by orchestrating multiple simulation environments in parallel, fully utilizing GPU resources for data collection and model updates. This parallelism results in orders-of-magnitude performance improvements compared to CPU-bound or single-threaded simulators, significantly reducing training times for deep reinforcement learning tasks.

A custom controller was implemented in Python, utilizing the gymnasium interface for environment setup and Stable Baselines3 for the reinforcement learning algorithms, including PPO, SAC, and their variants. The GPU-accelerated physics and multi-environment capabilities of Isaac Lab ensured realistic simulation fidelity and efficient policy convergence.

Figure 7: Nvidia Isaac Sim environment.

2.1.3.2 Training Process

The training process follows the widely used agent–environment loop, where the agent receives an observation, selects an action, and the environment returns a new observation along with a reward vector. In this implementation, the Proximal Policy Optimization (PPO algorithm was selected for policy updates. At each timestep, the actor network receives an observation vector comprising of proprioceptive data (joint positions, joint velocities, previous action signals, and current end-effector pose) alongside exteroceptive inputs (target end-effector coordinates and raw camera image). Based on these inputs, the actor produces continuous control commands (joint position commands) that actuate the robot's motors within the simulation multiple times per second. Simultaneously, the critic network estimates the state value given the same observation. Once an action is executed, an immediate reward reflecting task-specific objectives such as minimizing end-effector error or avoiding collisions is computed. Collected observations, actions, and rewards are then used to compute advantage estimates and surrogate policy gradients following the algorithm's clipped objective, which constrains policy updates to remain within an acceptable range to avoid sudden and unpredictable changes in action output. Policy and value network parameters are updated in minibatches over multiple epochs.

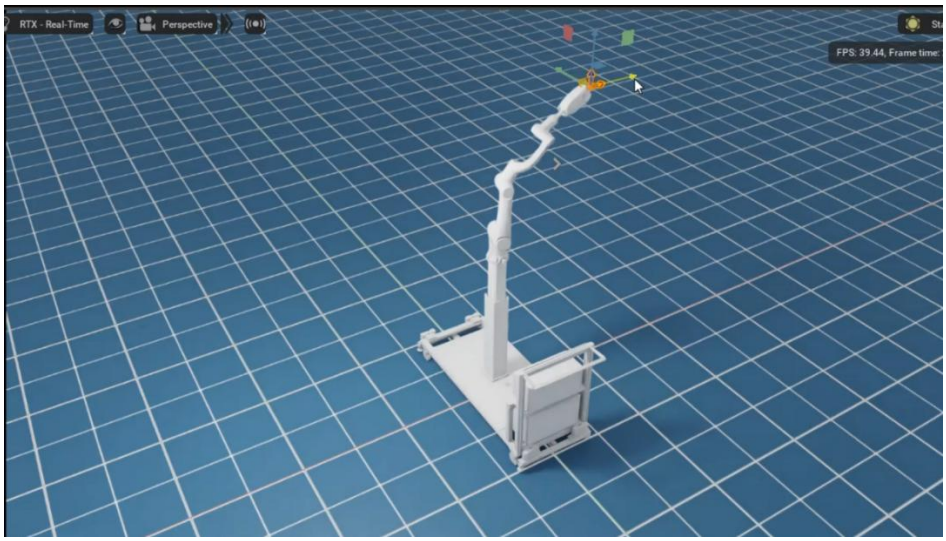

Figure 8: Trained agent in simple environment reaching dynamic target pose.

2.1.3.3 Environment Setup

A simple training environment was first constructed consisting of the RIR robotic manipulator and two sequential target poses within its working envelope. To encourage generalization of solutions, the reward function was kept simple and contained the following:

1. Positive reinforcement for reducing the Cartesian distance and orientation error between the end effector and the active target
2. Softer penalties for excessive joint velocities or accelerations to discourage unexpected motions.

Observations mirror this simplicity, including only the current and desired end-effector positions and orientations, together with the positions and velocities of all non-static joints.

Figure 9: Training session of RIR robot (No other cell objects).

Training was performed entirely within NVIDIA Isaac Lab, with 1,024 parallel simulation instances on a single NVIDIA GeForce RTX 3060 GPU. After systematically tuning the relative weights of the reward components, to not overpower the main objective of reaching the target, the policy converged reliably within approximately one hour of wall-clock time. The resulting controller consistently and smoothly guided the robot to each target in sequence, demonstrating the effectiveness of a streamlined reward and observation design. When exported and executed in Isaac Sim against novel target configurations, the same policy yielded accurate, fluid motions, demonstrating the effectiveness of this streamlined reward and observation design.

2.1.3.4 Insertion of airplane tank

To better fit reinforcement learning workflow to the IAI scenario, a high-fidelity virtual environment was constructed incorporating a detailed 3D scan of the airplane wing along with its supporting structure. Early experiments using the existing reward and observation schemas yielded unsatisfactory results—only about 40 % of training episodes successfully reached the two sequential targets, and policy performance on novel poses fell below 20 % after exporting to Isaac Sim. This resulted in an overhauled both on the reward and observation definitions but also on augmenting the sensory inputs to provide richer environmental feedback.

Initial testing was conducted, using an RTX-based ray-casting sensor for distance measurement and running 256 parallel environments due to limited computational resources.

Following an upgrade to an Nvidia H100 GPU the pipeline scaled to 2048 environment instances equipped with fully rendering RGB-D cameras, accelerating training dramatically.

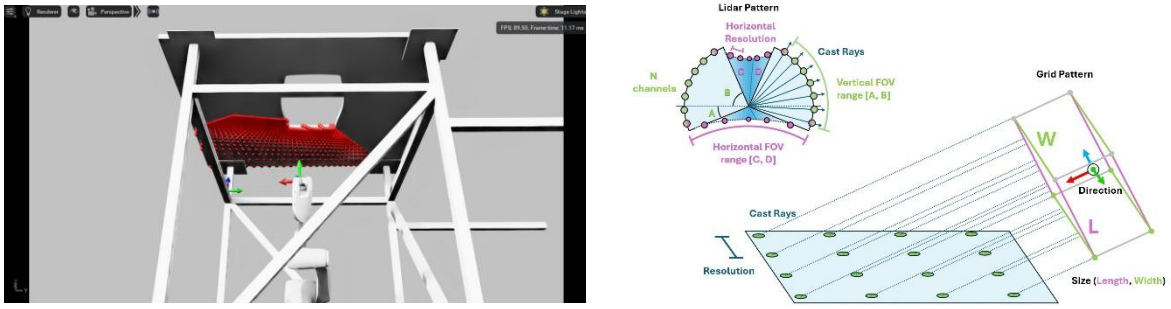

Figure 10: Visualized output of the ray-casting sensor (left) and casting techniques of the sensor (right).

The case environment introduction was done using an iterative approach. Training first on a minimal “box” deconstruction of the wing (initially open-top, then fully enclosed) enabled easier tuning of reward weights and camera configurations. Once the agent reliably reached all poses within the box, a 1.5×-scaled voxelized mesh of the wing was introduced to approximate the true geometry at a lower resolution. This intermediate stage further refined reward parameters and indicated some perceptual issues.

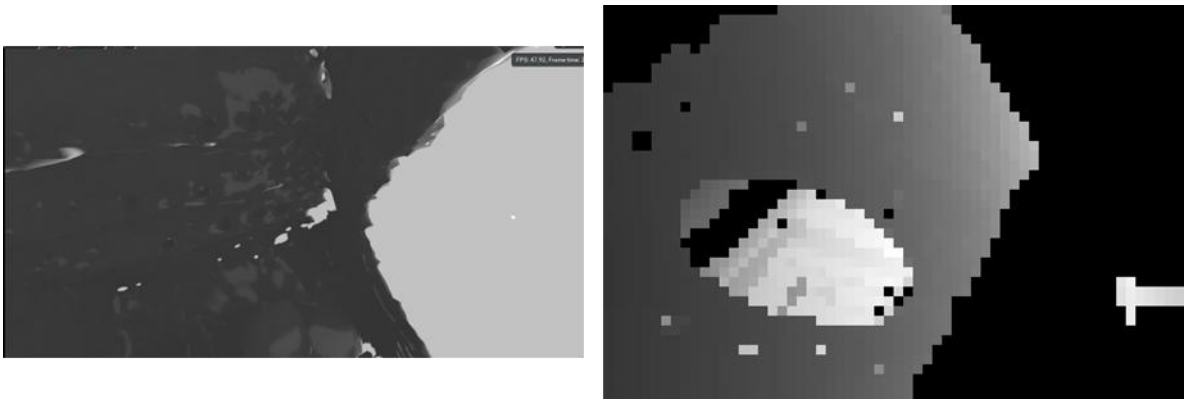

Figure 11: RGB (right) and depth (left) output of the camera sensor.

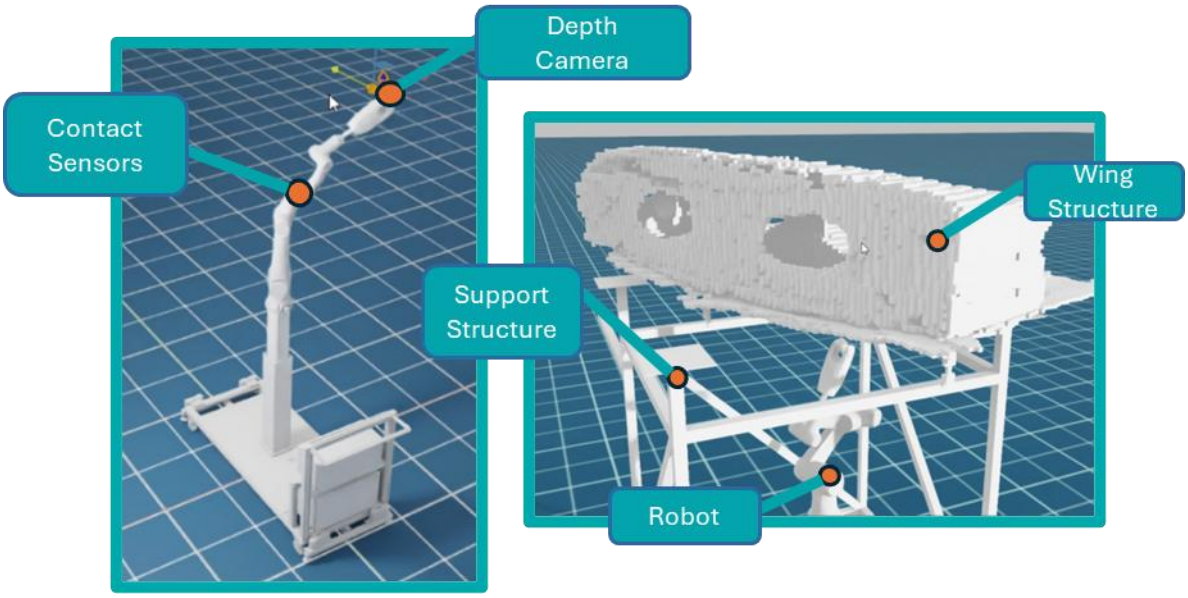

Figure 12: Environment setup.

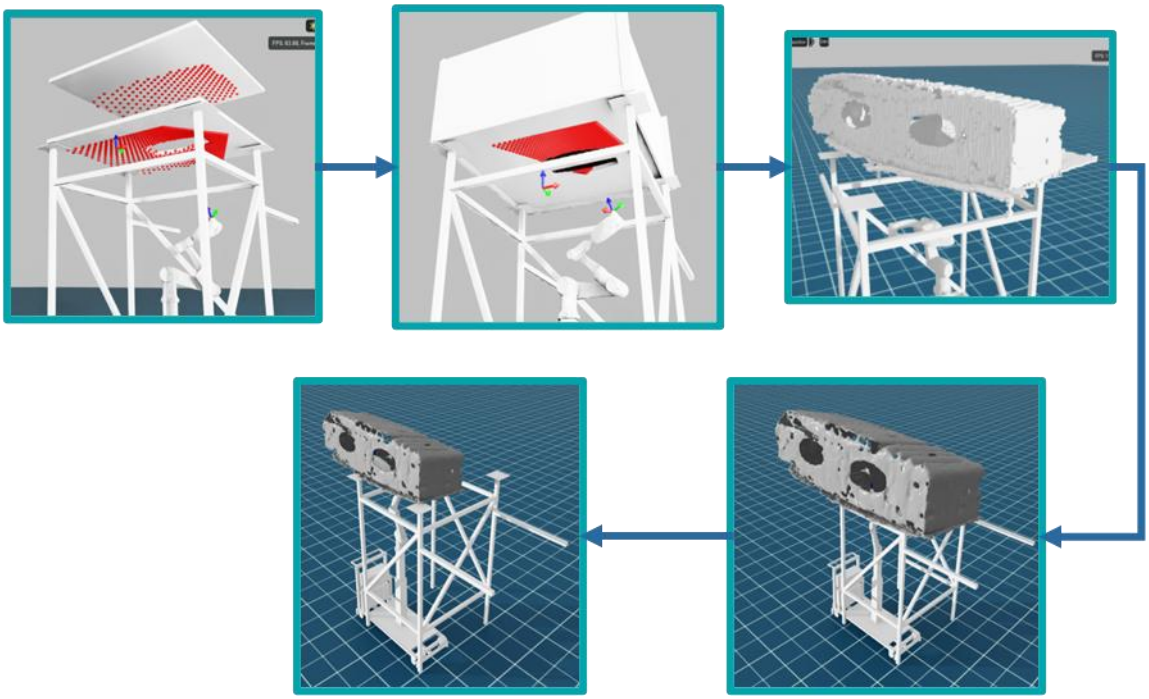

Figure 13: Training environment evolution changes.

Finally, the full-resolution, smoothed wing mesh replaced the intermediate models, revealing the remaining challenges that will be detailed in a later section. A full table of the final rewards can be found below:

Table 1: Final rewards description.

Reward Name	Function/Purpose
Target position/orientation	Minimizing tracking error
Fine-grained positioning	Stabilize target position tracking
Contact time & force penalty	Discourage collisions
Joint speed/Action rate	Stabilize movement
Termination Penalty	Discourages collisions

By progressively increasing geometric and sensory complexity—paired with hardware-driven parallelism—this staged approach ensured stable convergence at each level before advancing, laying a robust foundation for the final deployment on the real Aeronautics inspection task.

2.1.3.5 Challenges and Solutions

Initially there was a tendency for the agent to hover directly beneath the wing structure and aligning only on the x-y plane, exploiting a local minimum for the reward. The penalty for a collision was the discouraging factor for entering the wing. Since collision on a system like this should be avoided at all costs a new z-axis reward term was introduced. However, the collision penalty still dominated and did not allow the robot to reach most position that are further inside the wing.

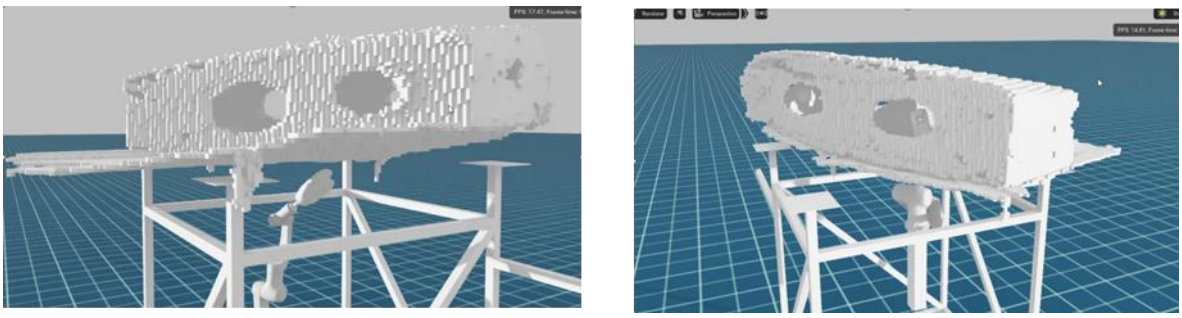

Figure 14: Agent exploiting local minimum (left) and side effect of adding z-axis penalty (right).

In response to this behaviour the robot starting pose and its relative position to the wing and supporting structure were adjusted heavily. With this change improved reachability in more challenging areas and revealed that the z-axis penalty was accomplishing the same result as the more general position reward and it was removed.

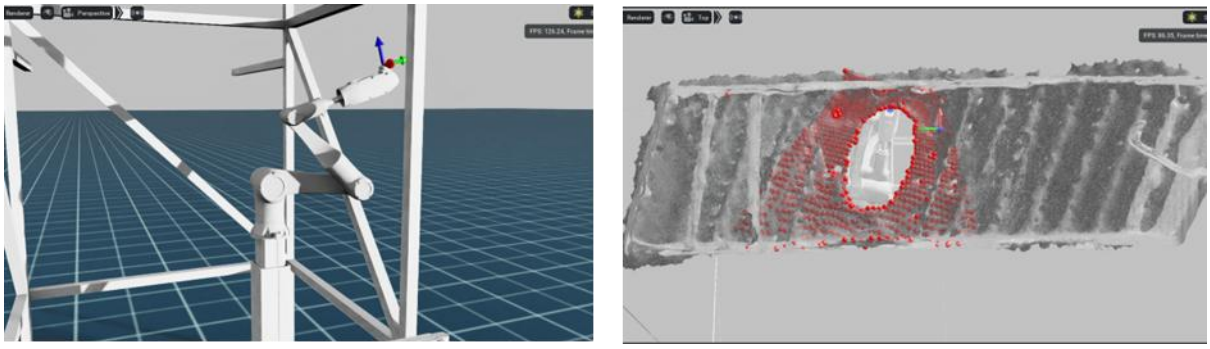

Figure 15: Adjusted robot starting position (left) and new alignment of the wing (right).

The agent would also momentarily graze objects to pivot more effectively. A new termination criterion was then put in effect that ends the training episode immediately after the total contact time with the environment has surpasses a predefined limit.

The most fundamental and critical challenge is the bridging of the simulation/training with the real environment and enabling on-site deployment. The robot's custom parameters (joint damping and stiffness), controller frequency, camera characteristics (intrinsic and extrinsic) were calibrated to match the physical hardware specifications. Domain randomization of robot starting positions, or environmental parameters also aids in a stable transition to the real world.

Matching 1:1 the simulation to the real world remains a challenge. Communication of the custom agent with the real actuators of the robot must be through a pipeline of technologies (ROS joint controllers, MoveIt!, TCP communication) and in combination of the precise tuning the virtual environment makes this task challenging.

2.1.3.6 Neural Network Desing

Two of the most prominent actor-critic network architectures were investigated to find a balance between computational performance and environmental representation.

The Recurrent Neural Network (RNN) design incorporates recurrent neural units to maintain an internal hidden state across timesteps during a training episode, enabling extraction of temporal connections in the agent's observation history. This memory buffer can improve exploration by correlating past collision events or depth ambiguities. However, the sequential nature of this network introduces additional compute overhead which can bottleneck high-frequency control loops. Moreover, the temporal correlation advantages delivered RNNs are

outweighed by the poor spatial feature processing and extraction at the end making them a unfit candidate for visual aided robot path planning.

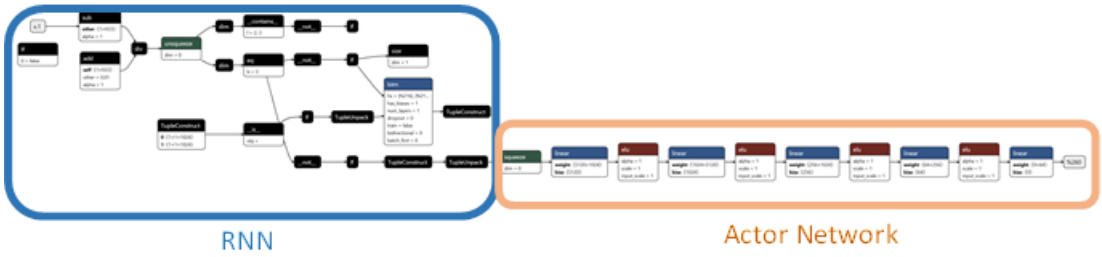

Figure 16: Recurrent Neural Network architecture.

The Convolutional Neural Network (CNN) architecture processes full-frame RGB-D inputs with a cascade of convolutional layers to extract spatial features (edges, local textures, and surface cues) that directly correlate with the wing geometry and support structure. These visual embeddings are concatenated with proprioceptive data (joint positions, velocities, and end-effector pose) before passing through fully connected layers for policy and value estimation. By leveraging GPU-accelerated convolution, this design achieves low latency per control step and high throughput when running thousands of parallel environments. On the downside, CNNs can require substantial VRAM when scaling to very high-resolution inputs and subsequently the camera image resolution could not surpass a 70x70 matrix when high parallelism needed to be maintained.

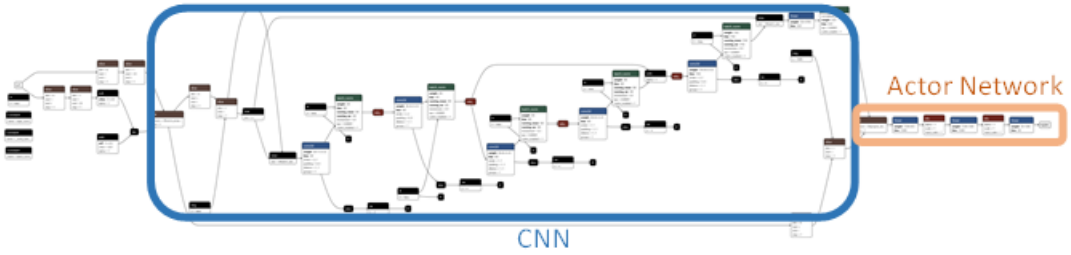

Figure 17: Convolutional Neural Network Architecture.

2.1.3.7 Results and observations

During multiple training sessions and after reward weight tuning, the CNN design outclassed the RNN in several key areas. After the training converged the former achieved significantly less episode terminations, indicating an increased success rate in a real-world scenario.

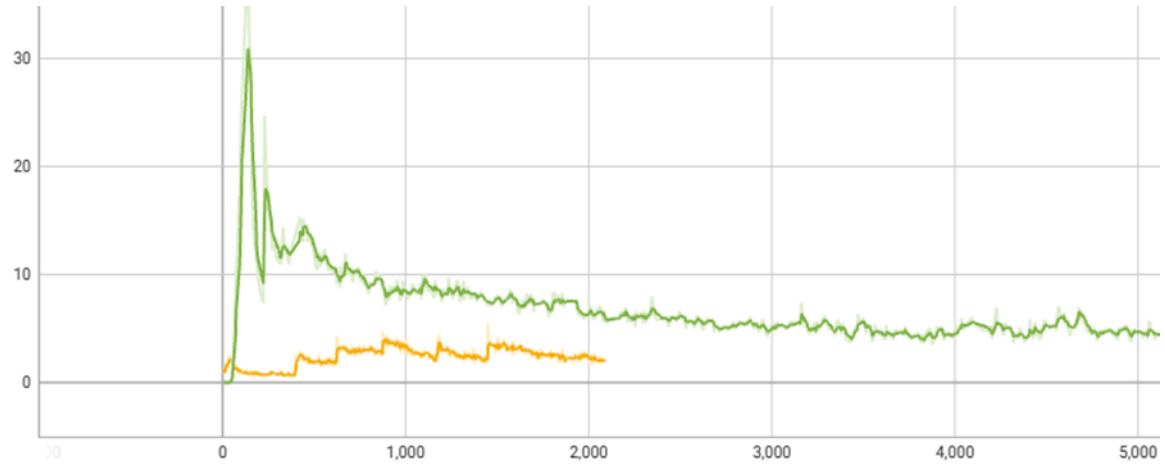

Figure 18: Termination timeout for RNN (green) and CNN (yellow).

Additionally, the cumulative reward score for the RNN network converged to a lower value, signifying underperformance, and took grater wall-clock time than a CNN architecture making it more computationally performant.

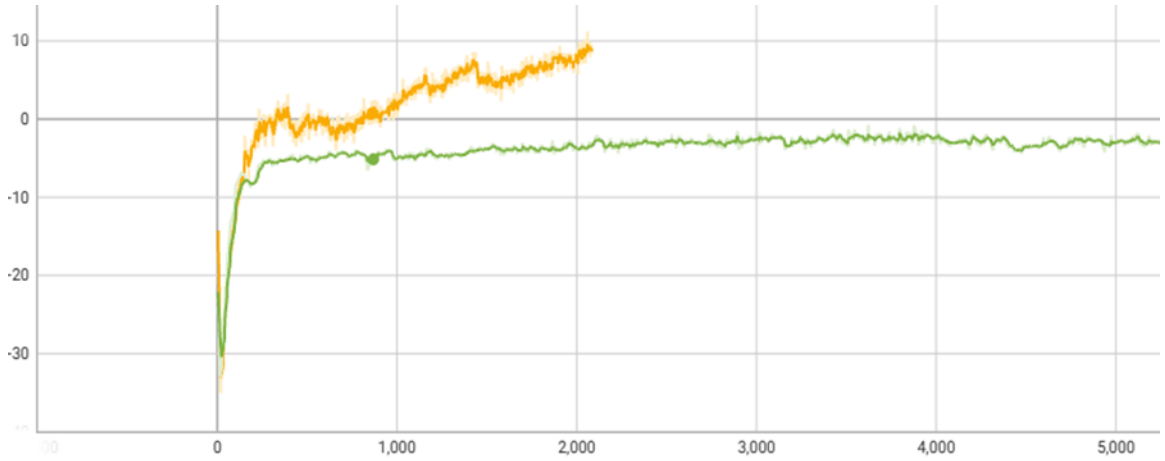

Figure 19: Mean Reward for RNN (green) and CNN (yellow).

2.2 Integration Progress in Pilot Applications

The AIDT module is integrated into the Converging Pilot lines, covering the entire lifecycle from Concept to Operation using the PDCA (Plan-Do-Check-Adjust) methodology.

It all begins with the Digital Twin Prototype, developed based on pilot scenarios, user requirements, and the overall architecture. This prototype is then updated with data from Research and Technology Organizations (RTOs) pilot leads, who perform physical experiments to gather valuable insights. This involves integration with CONVERGING Modules, specialized technological components that enhance the digital twin's capabilities. From the Digital Twin Instance, two main branches emerge: one leads to the Digital Twin Instance of the Industrial Demonstrator, which is part of the Converging solution demonstration on-site at pilot partners' premises. The other leads to the Digital Twin Instance of the Open-Pilot Demonstrator, designed for broader, more open-ended applications within research and

technology organizations. The integration process involves several steps and varies depending on the specific pilot line.

2.2.1 Automotive Manufacturing Pilot Case

In the Automotive Pilot Case, the AIDT integration is being tested on the realization of different concepts developed during the project. The technical development of the automotive pilot has focused on implementing CONVERGING solutions in the mobile polishing system as well as in a multi-robot configuration system.

The focus has been on testing the AIDT module against this configuration to bring the concept to operation. The use of AIDT virtual commissioning tools has been introduced to automotive stakeholders, improving their competency in building digital twins and using the tools provided by the AIDT module. This involves concurrent engineering practices for simultaneous improvements between digital and physical execution.

In the automotive pilot line, the AIDT integration is being tested on various concepts developed during the project. The technical development of the automotive pilot has focused on implementing CONVERGING solutions in the mobile polishing system and in a multi-robot configuration system.

The emphasis has been on testing the AIDT module within this setup to bring the concept into operation. The use of AIDT virtual commissioning tools has been introduced to automotive stakeholders, enhancing their skills in building digital twins and utilizing the tools provided by the AIDT module. This involves concurrent engineering practices for simultaneous improvements between digital and physical implementations.

The pilot line has demonstrated the AIDT functionalities, including the virtual design and representation of the automotive shopfloor at different levels of detail.

Furthermore, automotive pilot line started with the conceptual phase where simulation was used for designing the early system keeping in mind the industrial and research aspects and tackling different aspects of implementation challenges that arise from tests onsite shown in Figure 20.

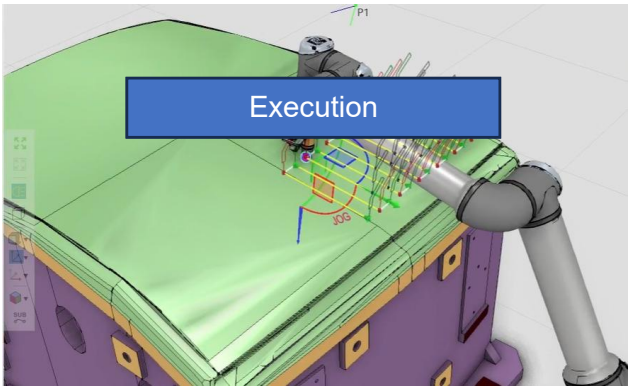

Figure 20: Initial Stationary robot system.

Until r4.8, polishing tasks were not supported, and the VC4.0 platform was extended with features compatible with the polishing tasks to support the Automotive pilot line.

Furthermore, collaborating mobile polishing robot virtual design (Figure 21), was carried out in VC4.0 (r4.9), which helped in evaluating the mobile polishing robot solution in a collaborative environment.

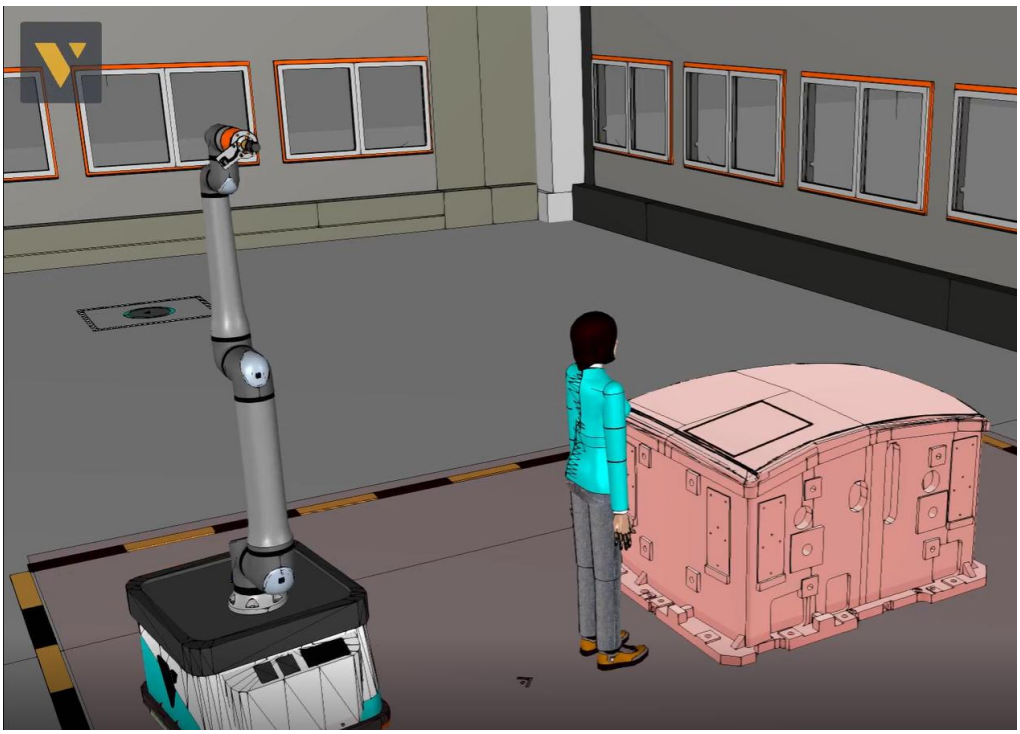

Figure 21: Mobile Human-Robot Collaborative Polishing Robot System.

2) Full support of Mobile Polishing robots in the form of visualization to enable realistic simulations (robot shape, kinematics, position of the end effector, robot pose).

Following the development of concepts, programming, and testing on-site of the previous solutions. The final Mobile Polishing Robot solution was developed by Tecnalia, shown in Figure 22. Each component of the solution was developed in the CONVERGING simulation library (r4.10), the end effector component (Figure 23), the mobile robot component (Figure 24).

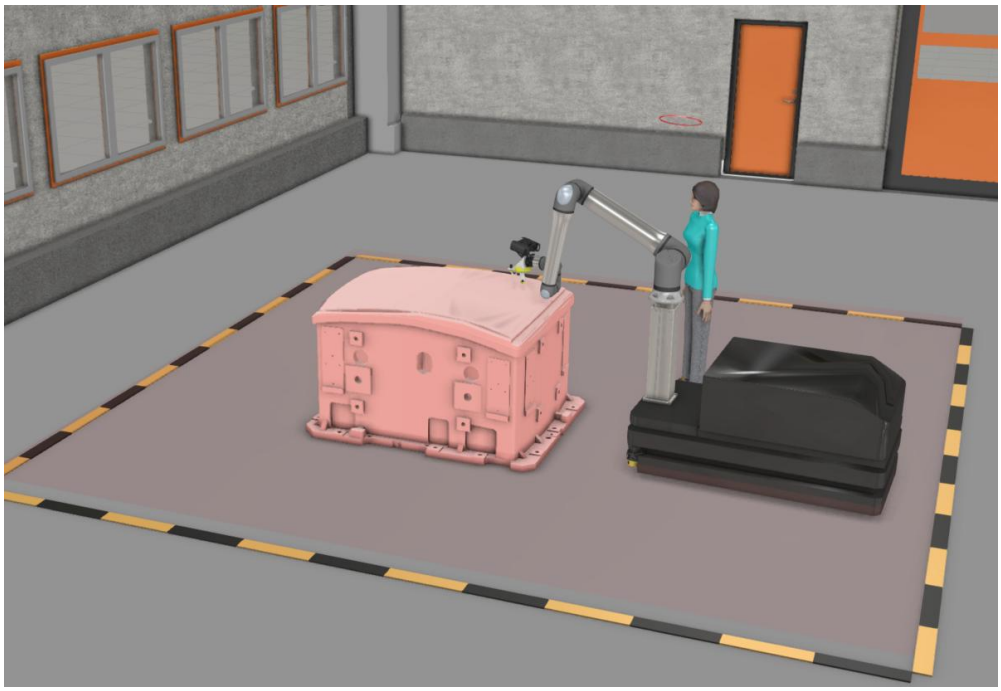

Figure 22: Mobile Polishing Robot (Tecnalia) in VC4.0 (r4.10).

Figure 23: Mobile Polishing Robot End Effector Design.

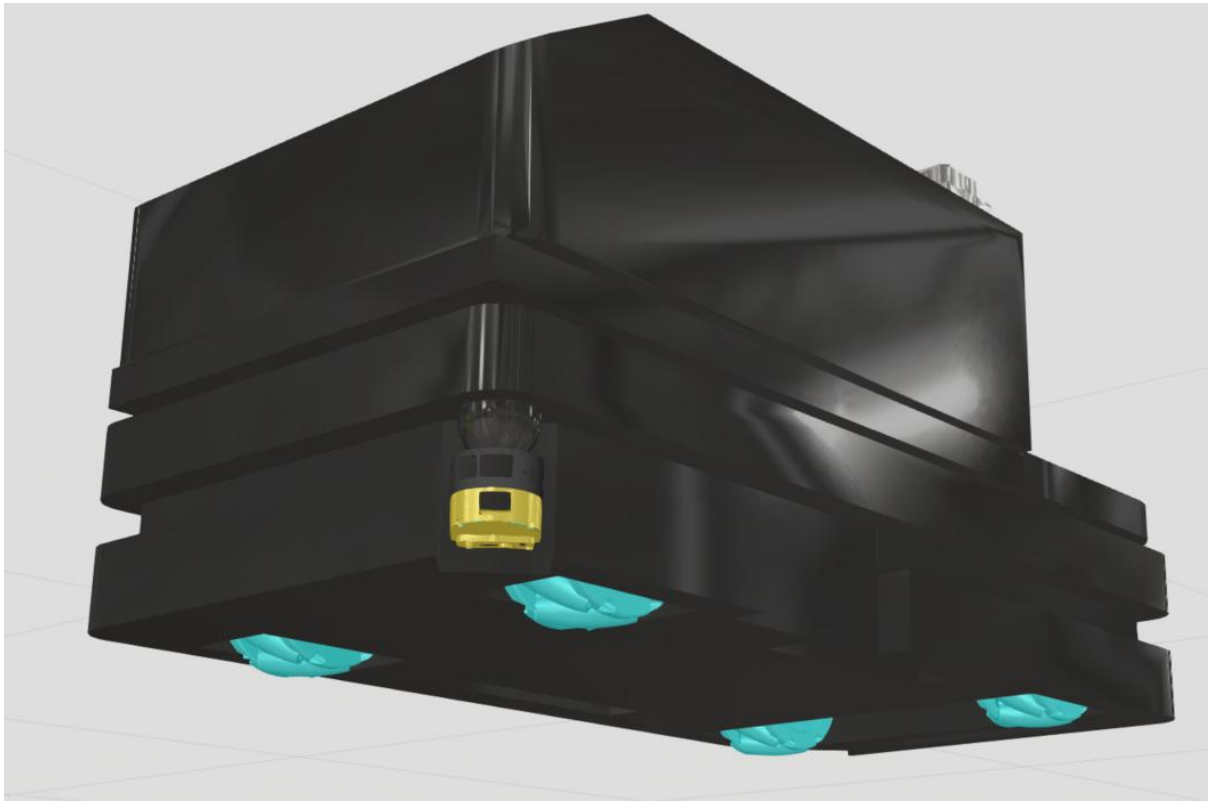

Figure 24: Mobile robot of the Mobile Polishing Robot.

3) Navigation capabilities for mobile robotic resources

Following the development in mobile robots' resources, the navigation capabilities need to be developed in the AIDT module, following the functionality requirements of AIDT. The mobile robot controller (Figure 25It is developed for these navigation capabilities.

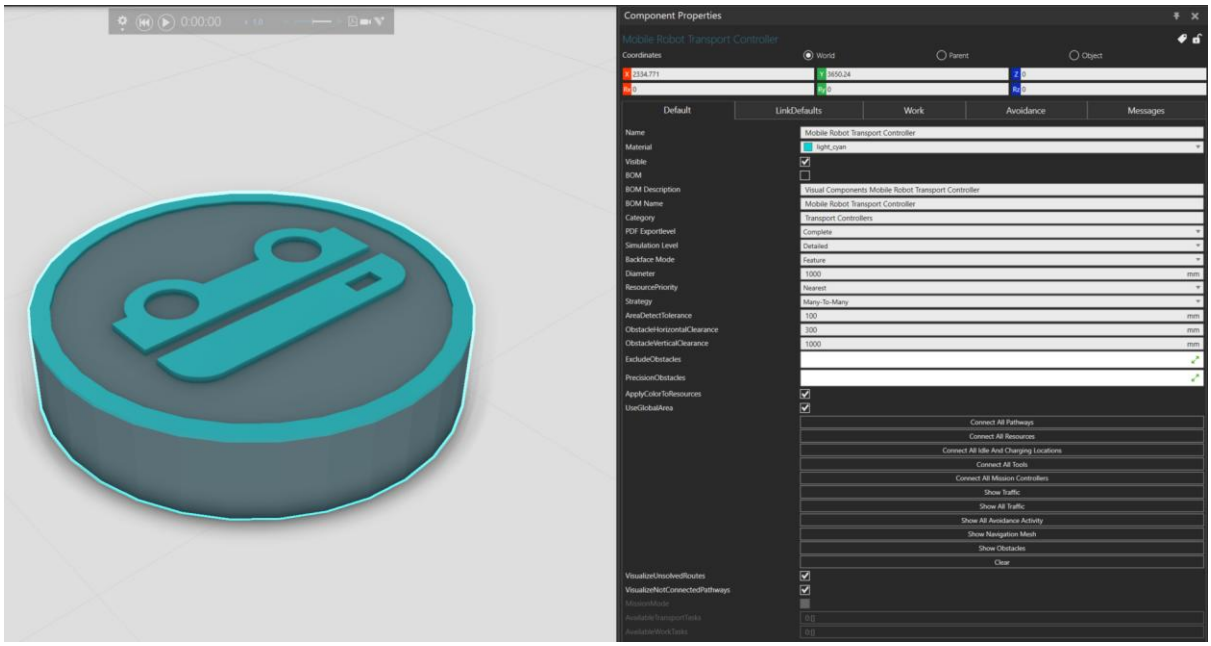

Figure 25: Mobile Robot Controller providing access to navigation capabilities VC4.0 (r4.10).

The mobile robot controller navigation considers obstacles and dynamic changes in the environment. This means mobile robots can navigate around each other and around objects without stopping unnecessarily, which is crucial for the use of a mobile polishing robot as it will be part of a complex shopfloor environment. Figure 26 shows the user interface of the navigation capabilities.

Figure 26: Major capabilities for navigating the mobile robot in AIDT.

2.2.2 Additive Manufacturing Pilot Case

The Additive Manufacturing pilot places a significant emphasis on ergonomics and represents a medium payload collaborative robot, ensuring that the digital twin technology supports human-centric manufacturing.

The AI Digital Twin (AIDT) module in the Additive Manufacturing Pilot Line has focused on ergonomic assessments using the RULA (Rapid Upper Limb Assessment) and REBA (Rapid Entire Body Assessment) methods. These methods are crucial for evaluating the risk of musculoskeletal disorders in the workplace.

2.2.2.1 RULA and REBA Components:

- RULA: This method assesses the upper limbs, including the arms, wrists, and neck. It evaluates the posture, force, and repetition of tasks performed by the upper body. The assessment involves scoring the posture of each body part, considering factors such as the angle of the joints, the amount of force exerted, and the frequency of movements. The scores are then combined to provide an overall risk level, helping identify areas that require ergonomic intervention to reduce the risk of injury.
- REBA: This method assesses the entire body, including the trunk, neck, legs, and arms. It evaluates a broader range of tasks and postures compared to RULA. The assessment involves scoring the posture of each body part, considering the angles, forces, and movements involved. The scores are then combined to provide an overall risk level, helping identify high-risk tasks and postures that require attention to improve workplace ergonomics.

Implementation and Visualization

The RULA and REBA methods have been added to human simulation models in the CONVERGING Simulation Library. These components are accessible through the online eCatalog and are valid for Visual Components 4.0 (VC4.0) from version r4.10 onwards.

The ergonomic components have a dynamic visualization and scoring system. This allows for evaluating ergonomics with respect to pose at different timestamps, and the data can be stored and extracted for performing big data analysis related to ergonomics evaluation.

The visualization of the scores is designed to be user-friendly, with different color coding to indicate the ergonomics score shown in Figure 27. This helps to observe the ergonomics score intuitively and make design changes as required.

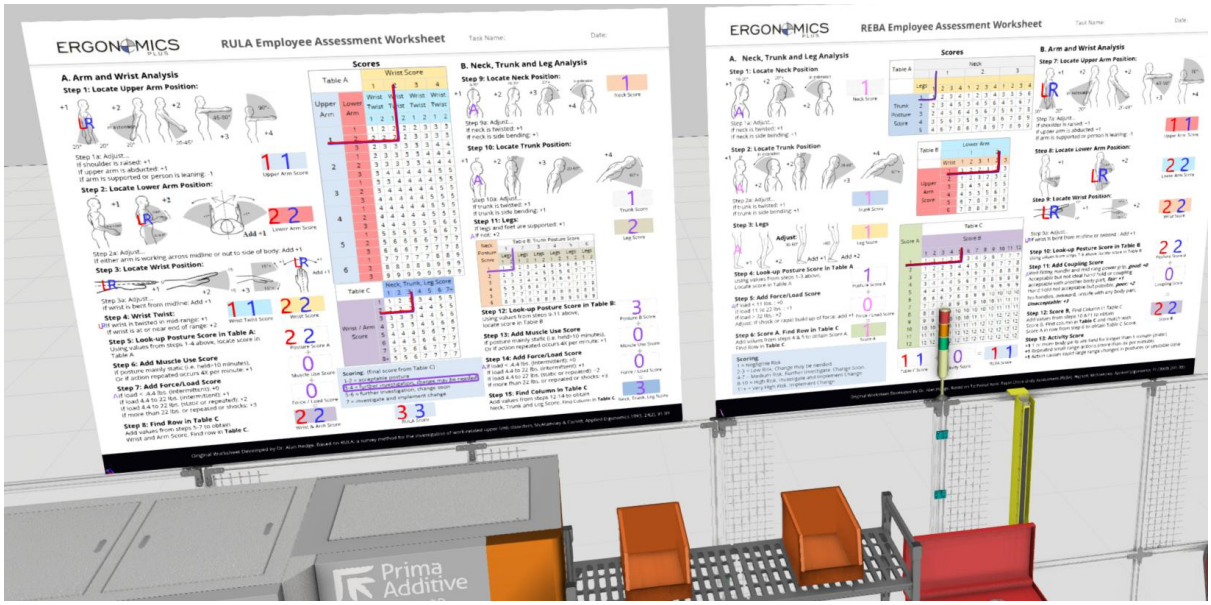

Figure 27: REBA and RULA in VC4.0(R4.10).

2.2.2.2 Real Human Pose Data Integration

Research has been conducted on how to represent real human pose data in a digital human model. The ergonomics components are designed so that if data from the real environment is captured about human movements, the ergonomics could be evaluated and demonstrated, as shown in Figure 28.

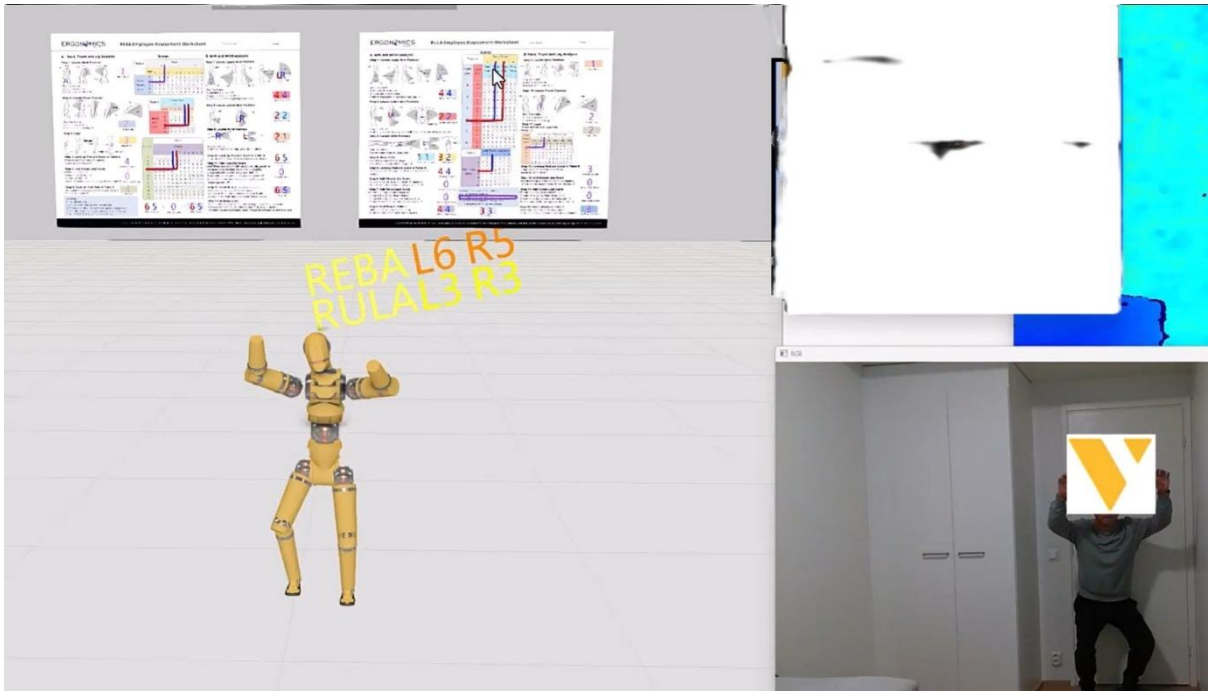

Figure 28: Real human poses live interaction and dynamic RULA, REBA score.

Furthermore, for the Additive Manufacturing Process, a detailed ergonomics assessment is required, for example, the use of the human hand and generating big data ergonomics based on it. This capability has been developed in VC4.0(r4.10), depicted in Figure 29.

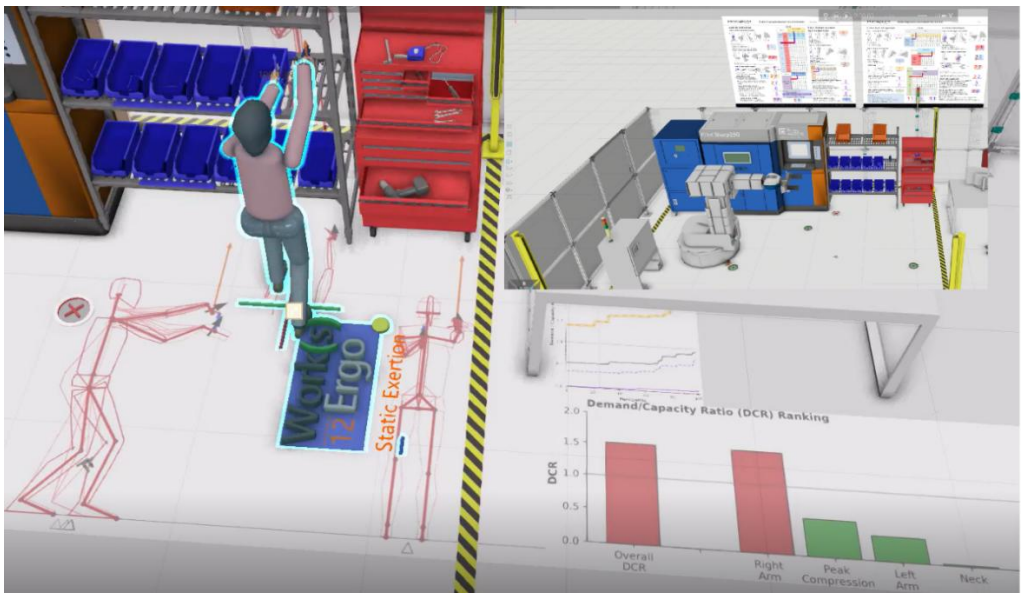

Figure 29: Refined and detailed ergonomics calculations using Big Data Ergonomics Platforms and VC4.0 (r4.10) in the Additive Manufacturing Pilot Line.

The Additive Manufacturing pilot's focus on ergonomics ensures that the digital twin technology supports a safer and more efficient workplace by continuously integrating user requirements and feedback from physical experiments and industrial partners.

2.2.3 White Goods Manufacturing Pilot Case

In the White Goods pilot line, the focus of the AIDT module has been on representing complex humanoid robots shown in Figure 30 from Kawada in the environment and advancing the ease of programming of such robots in digital environments.

The CONVERGING Simulation Library has integrated support for the humanoid-type solution, which can be leveraged for engineering solutions with such robot systems, utilizing concurrent engineering practices that have been proven for industrial robot arms using digital twin solutions.

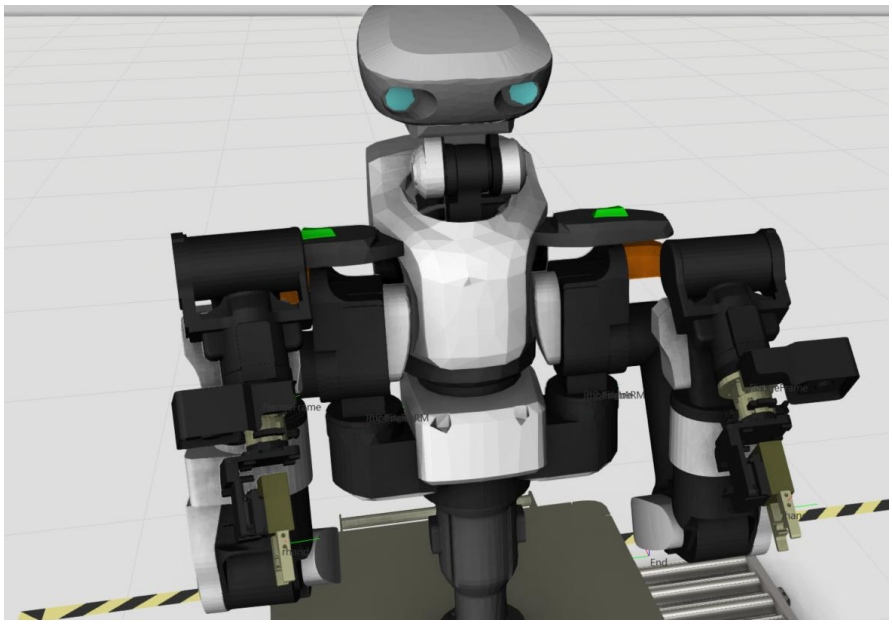

Figure 30: Kawada Humanoid Robot in VC4.0.

Using the digital humanoid robot, several CONVERGING White Goods pilot line scenarios were developed and studies, as shown in Figure 31.

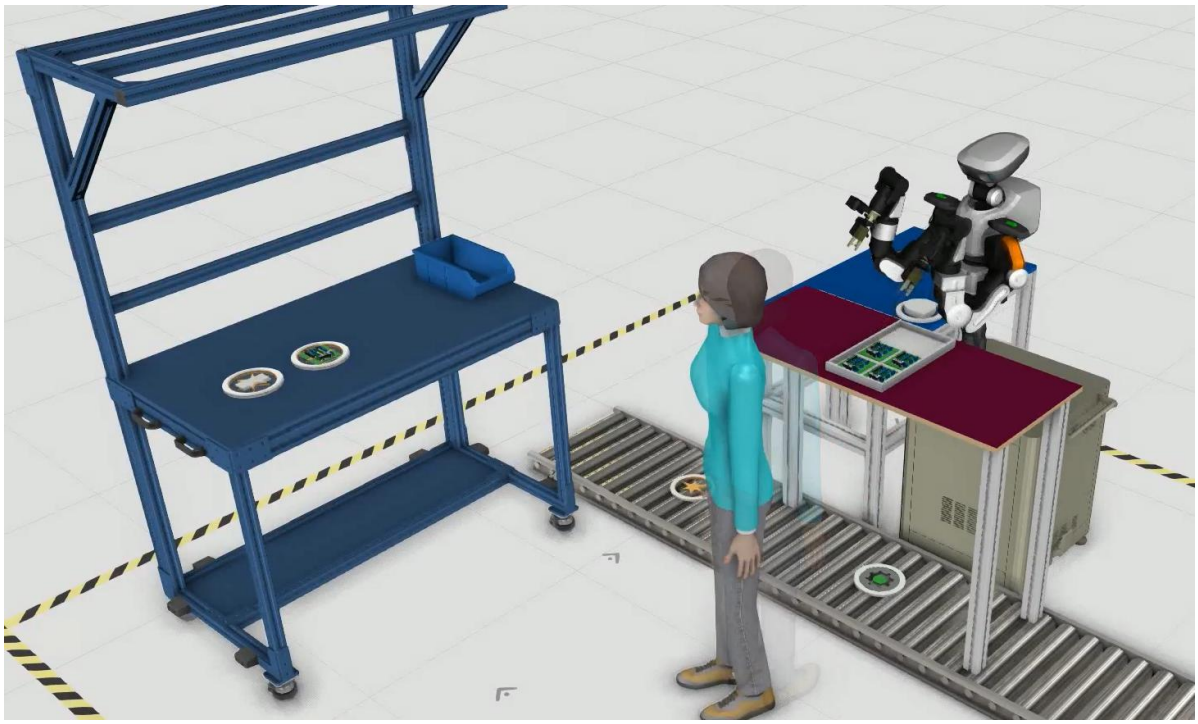

Figure 31: The initial converging scenario for white goods pilot line described in D4.1 and D4.2 depicted in VC4.0.

Following the developed scenario, PILZ carried out the safety design (Figure 32) of the scenario demonstrating.

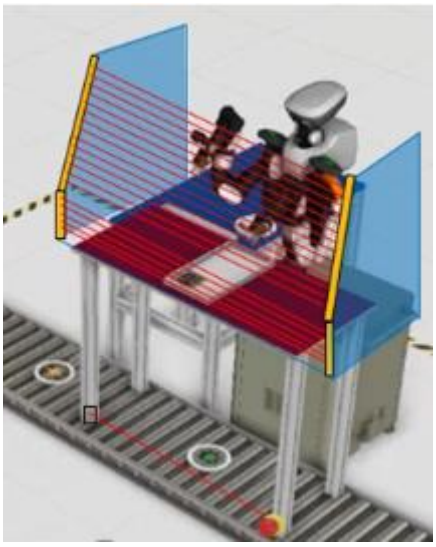

Figure 32: Conducting safety design for the scenario in VC platform.

During this period, the focus was on updating the digital representation of the humanoid robot to synchronize both arms for simultaneous pick-and-place operations in VC5.0, which introduced a new robot controller and Python 3 API in this version. The focus has been on establishing API connectivity between VC and the humanoid robot based on Python 3.

To focus on the behaviour modelling of the robot and close the gap between real and simulated behaviour in the VC 5.0 platform, the robot has been tested in several different existing digital layouts available in the VC platform, shown in Figure 33.

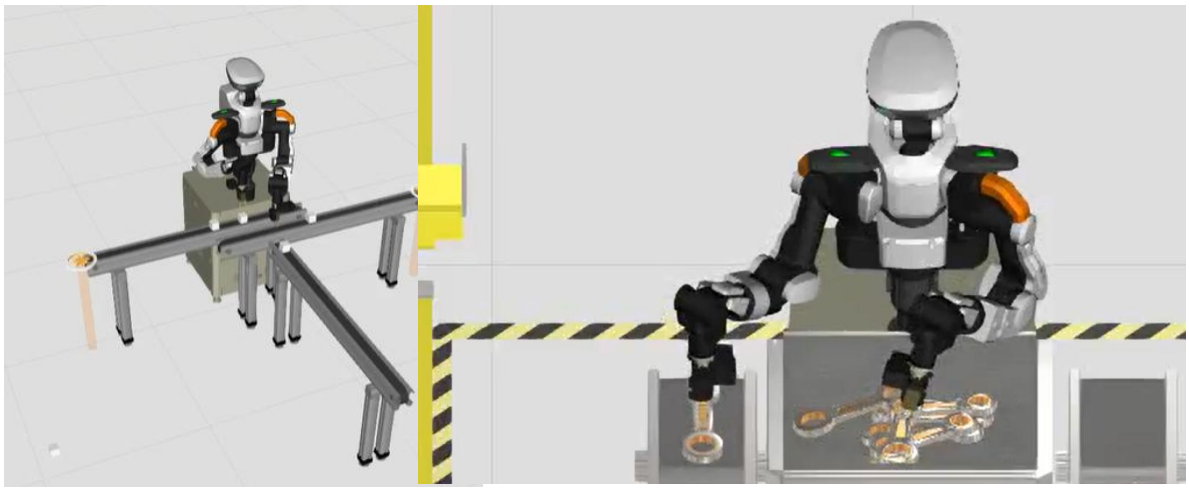

Figure 33: Moving objects pick and place in a conveyor layout (left) and simultaneously pick and place in a vision-based bin picking layout (right).

The next step is to finalized the integration of the described functionality in the white goods pilot line in the WP7 pilot integration phase.

2.2.4 Aeronautics Manufacturing Pilot Case

The Aeronautics pilot case focuses on developing a dynamic digital twin that integrates real-time sensor data, particularly in complex environments such as fuel tanks and robot path planning for a customized robot system comprising linear axes and a custom robot arm.

In addition to the complexity of the working space inside the wind, the information to manipulate the robot inside the wing is provided in the form of point clouds.

As introduced in the section 2.1.1, support for handling point clouds has been extended in VC 4.0, as detailed in previous deliverables (D4.1 and D4.2) and integrated into VC 5.0. However, leveraging the robotics capabilities in VC has required adding information—using a voxel-like approach—to the point cloud that aids in robot path planning and collision detection.

2.2.4.1 Setting up the environment

The configuration of the environment in VC follows an intuitive approach that begins with setting up the work environment and the robot. The UI interface (Figure 34) provides the functionalities required so the operator can handle the tasks.

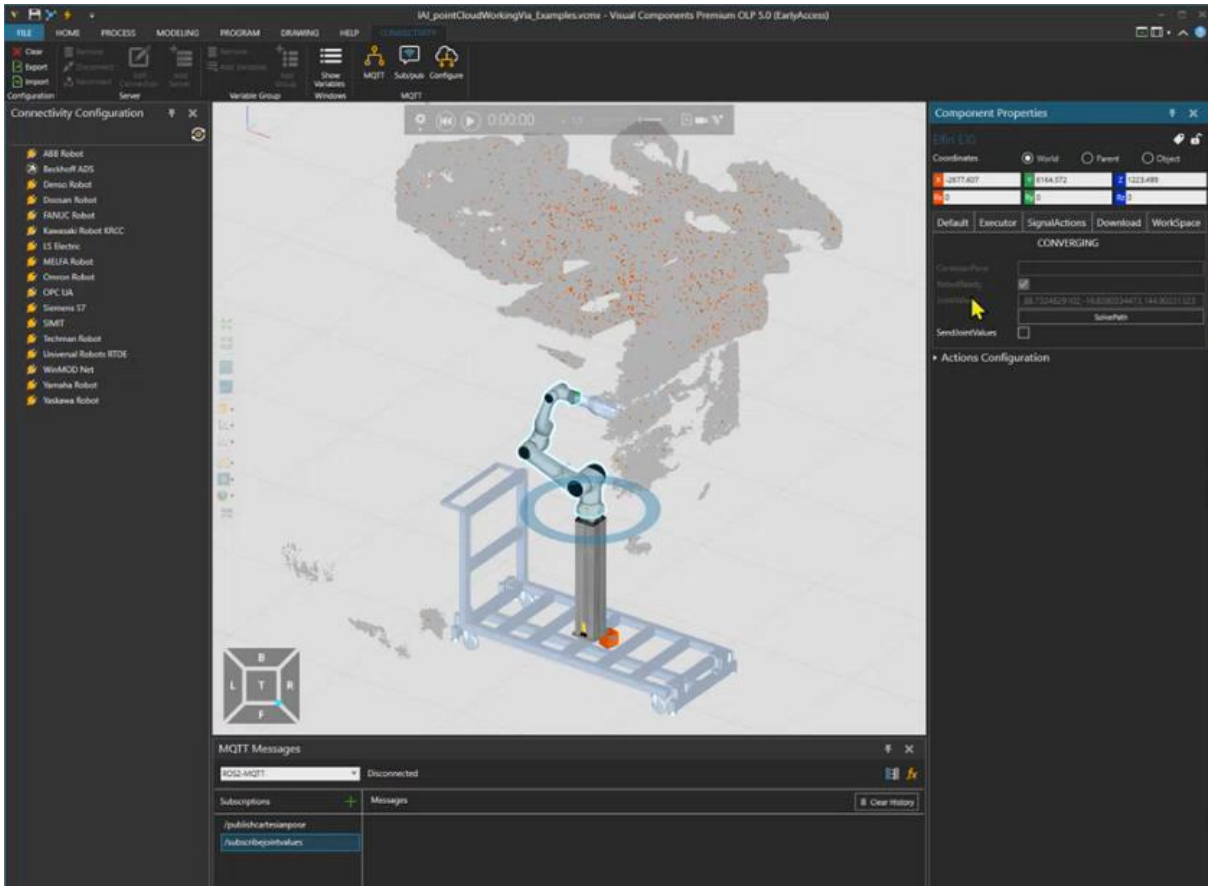

Figure 34: Setting up the Aeronautics Pilot Case environment in VC platform.

2.2.4.2 Point cloud collision detection

Point clouds provided by the PAM are imported into VC. Unlike other CAD formats traditionally used in the engineering process, this adds extra complexity to the trajectory calculation using VC solvers and colliders, which is why a voxel-like approach has been implemented.

Voxel-like approach means turning raw, unstructured point clouds (a set of 3D points) into a structured volumetric map made of small cubes called voxels (3D pixels). Each voxel represents a tiny cell of space and stores information (e.g., “occupied,” “free,” or a distance to the nearest surface). This enables spatial reasoning—such as collision checking, planning, and tracking—to be performed much faster, more robustly, and with greater ease than when working directly with raw point clouds.

A new simulation component has been developed for integrating this approach. This component generates voxels to define limits and check collisions. The component is deployed with a UI to access its capabilities. (Figure 35)

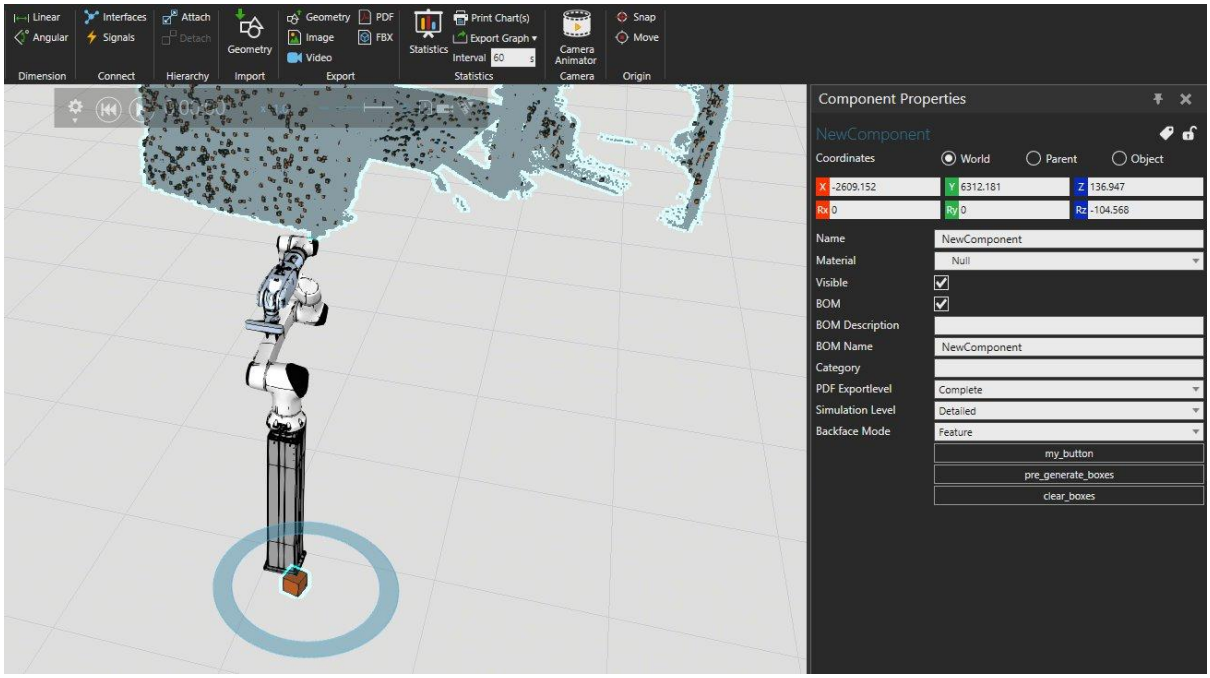

Figure 35: Screenshot of the voxel generation component with the UI to configure it.

2.2.4.3 Trajectory Calculation

Solver and colliders are used in trajectory calculation and robot path optimization. During this period, the robotic solver and colliders have been improved, making them available in VC 4.10 and VC 5.0. The robotic solver is accessible through VC UI and resolves common trajectory issues across different path types.

Capabilities provided by the robotic solver include detection and fixing of collisions, singularities, joint limit violations, and reachability problems. It works on process paths, search paths, and intermediate motions (via paths). Capabilities are configurable through the UI (Figure 36). To integrate into CONVERGING framework, as detailed in the section 2.1.1.2, a VC interface has been developed and integrated within VC 4.0 (r4.10) and extended into VC 5.0.

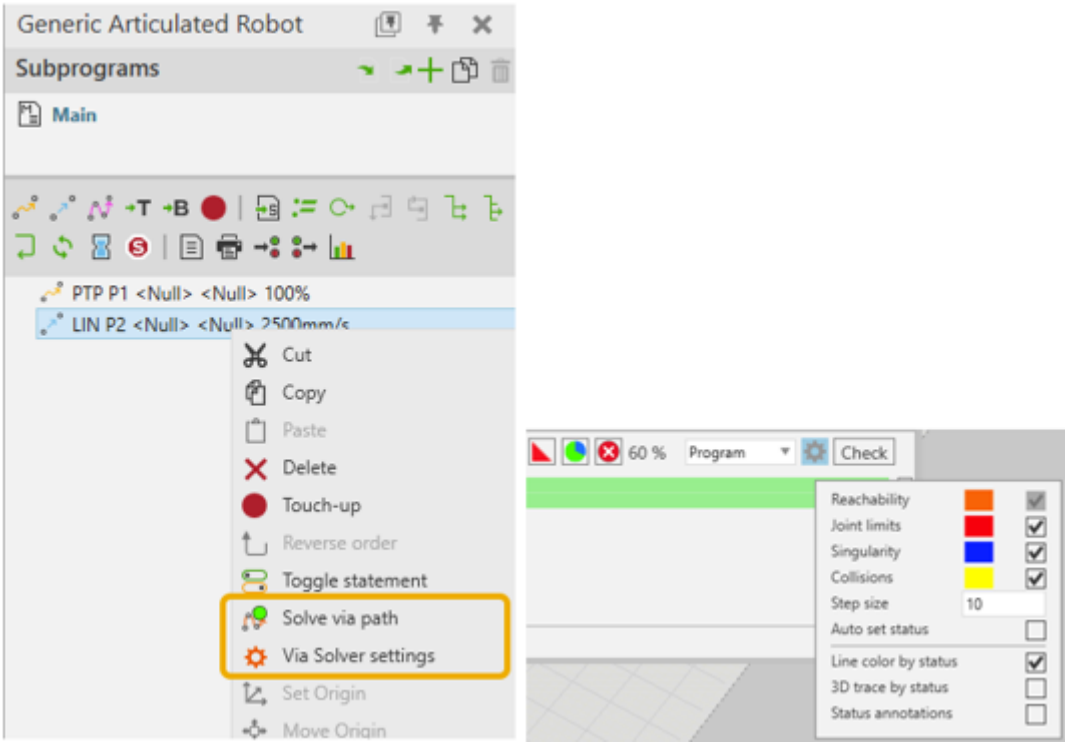

Figure 36: UI for path planning configuration.

2.3 Next Steps

In D5.3, the final steps towards the delivery of the AIDT functionalities described in D2.1 and D2.2 were presented. During the next project period, the work will be focused on fine-tuning the functionalities while deploying the pilots in line with the work in WP7. Additionally, real-time multi-axis robot path planning for confined spaces (IAI) will continue to be finetuned to achieve more accurate and robust results. The developers have already started the migration of the IAI demo to ROS 2, to make use of planners like cuRobo which take advantage of GPU acceleration, combined with the RL and graph approach described above. Any additional results will be reported in WP7.

3 Integration & Communication Architecture – Final Version

The goal of this section is to provide an overview of the final version of the integration and communication architecture implementation. The presented developments took place in the framework of T4.4 “Open integration and communication architecture for reconfigurable production – deployment”, which was responsible for the design and deployment of the reference architecture developed in T2.4 “Multi agent AI resources interconnectivity - Reference architecture for industrial take-up”. In addition, the materialization of the Open Flow based software platform has been completed, by finalizing the implementation of the related connectors. The aforementioned, AI Station Controller based platform, will be used as the backbone of the CONVERGING software system in terms of orchestration and information exchange. The CONVERGING reference architecture is documented in D2.2 - Design of the Reference Software Architecture.

3.1 Reference Architecture Implementation Updates

This section provides on overview of the process followed to develop and deploy the reference architecture. The methodology addressed the development and integration of all modules and module groups, including the Big Data Pipeline. It starts from initial prototype development to integrated testing with specific modules and finally towards the deployment to the Open Pilot Case environments. Figure 37 shows an outline of the different steps and stages.

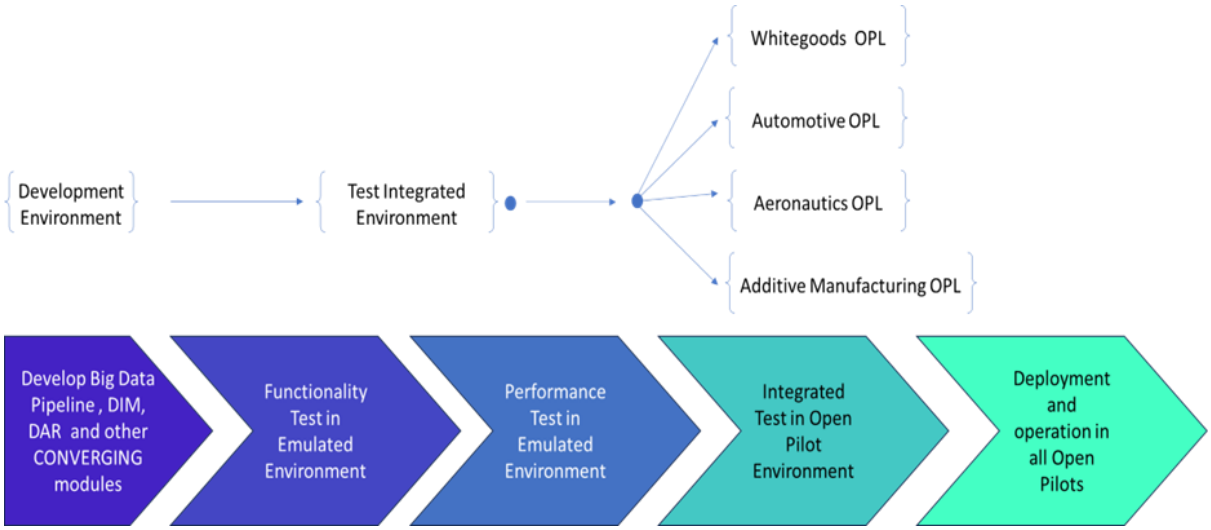

Figure 37: Big Data Pipeline Integrated architecture development approach

The first step was to develop the individual modules of the Big Data Pipeline as well as the other CONVERGING modules. This step took place concurrently with all CONVERGING developers, developing and testing individually their modules in their development environments.

During the next step partners shared module prototypes and created Integrated Test environments. The CONVERGING private Docker Repository was used to exchange software modules and docker-compose.yml files were used to create the integrated testing environments. A docker-compose.yml file is a configuration file used by Docker Compose, a tool for defining and running multi-container Docker applications. This file allows developers to specify the services, networks, and volumes that an application will use in a simple and declarative manner.

These integrated test environments were used to both test and verify functionality as well as to test and measure performance, in scenarios that replicated real world requirements and load. In addition to the CONVERGING docker repository, a dedicated GitHub organization, with private repositories has been created and used to exchange information and settings between the CONVERGING Consortium members. The tools used to support the integration are presented in dedicated sections.

The next step the different modules, started to be integrated into the Open Pilot Environments where different software module groups were also integrated and used with the innovative CONVERGING hardware.

Additional information on the CONVERGING Open Pilots are available in CONVERGING Deliverable D7.2 “Open pilots - Operating status and execution report”.

3.1.1 Big Data Pipeline Communication and Integration

The Big Data Pipeline integrates Data in Motion (DIM) for real-time streaming and Data at Rest (DAR) for persistence and analysis. Together, they enable seamless data flow, interoperability, and analytics across all CONVERGING modules

The Data in Motion (DIM) module address real-time data exchange between Human-Robot Collaboration (HRC) agents. It captures, fuses, processes, and streams information using Apache Kafka as its backbone for scalability and fault tolerance. DIM supports multiple input sources (ROS, MQTT, databases, files, AISC) and sinks data into DAR for persistence. This ensures low-latency, reliable, and contextualized data flow across CONVERGING modules.

In particular, DIM captures information via three different input methods:

- Via **DIM ROS streamer.** DIM ROS Streamer is a highly performant and configurable connector that captures data from ROS topics, converting them into JSON.
- Via **AISC Kafka connector**. AISC, who is responsible for orchestrating the overall process between different software modules. AISC can produce and stream real time information about the overall process execution, including information on the when each task and action started and stopped and their duration. DIM uses this information to calculate schedule execution specific statistics as well as aggregated statistics over multiple executions of the same schedule.
- Via **DIM CDC Streamer.** The DIM CDC Streamer utilizes Change Data Capture on connected persistence storage systems to efficiently capture, fuse, process and sink changes as events.

The Data at Rest (DAR) module is a core component of CONVERGING’s Big Data Pipeline. It is built on a microservices-based architecture to enable scalable and secure management of heterogeneous industrial data. DAR has two main elements: the Data Framework, which ensures efficient persistence, processing, and analysis of big data; and the Asset Administration Shell (AAS) Framework, which standardizes interoperability among assets using IDTA-compliant sub models. Together, these components facilitate seamless data sharing, transformation, and normalization, supporting predictive analytics, KPI visualization, and decision-making across all use cases.

The AI Digital Twin (AIDT) module creates virtual representations of shop floors and assets to support simulation, planning, and AI/ML integration. It includes libraries for ergonomics, path planning, and human-robot collaboration. AIDT enables testing, optimization, and predictive analysis in a safe digital environment. It bridges real and virtual data, enhancing decision-making and pilot case execution. AIDT is described in more detail in section 2.

The following subsections summarise the connections between the Data Pipeline via the DIM module to the other CONVERGING DIM modules in the CONVERGING OPLs as these are part of the integration and communication architecture.

3.1.1.1 Additive Manufacturing Pilot Case

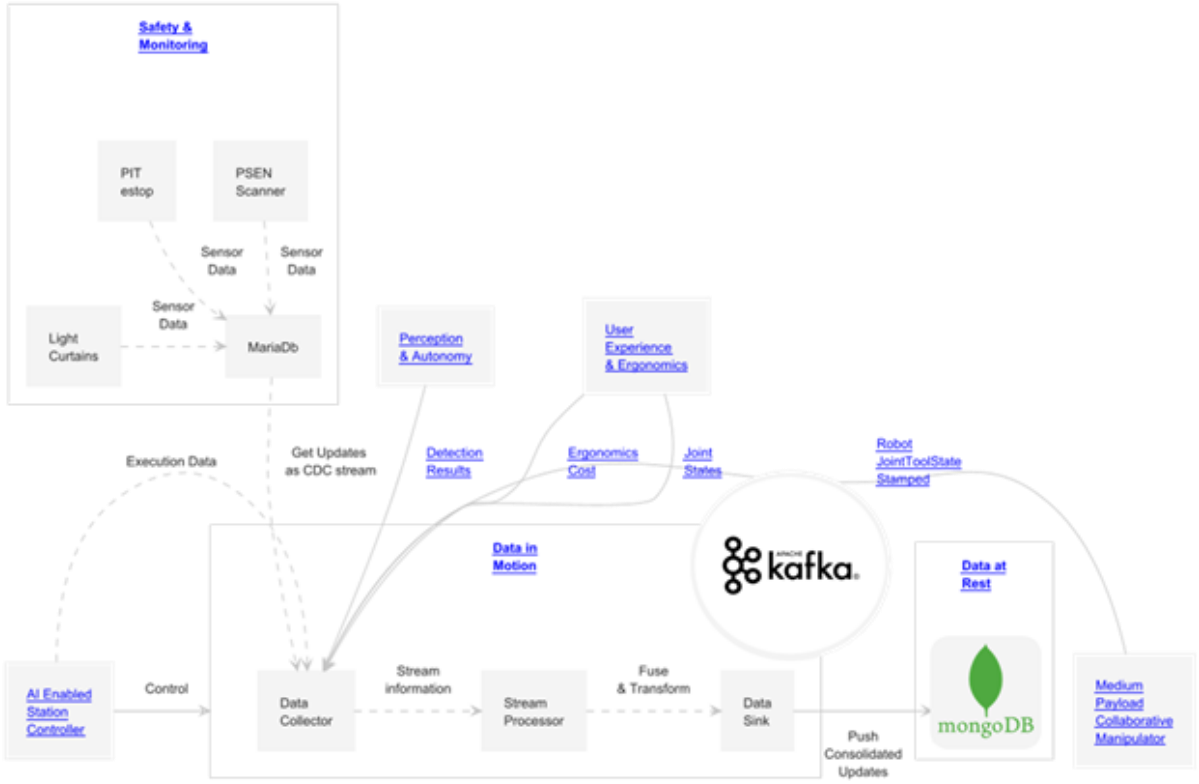

Figure 43: Additive Manufacturing - Big Data Pipeline

In the Additive Manufacturing Pilot case OPL the Big Data Pipeline collects data from multiple modules. Figure 43 shows a diagram that focuses on the DIM capturing streaming, fusing, transformation and sinking process in the Additive Manufacturing Open Pilot Line. the Medium Payload Collaborative Manipulator, timestamped robot joint and tool states are recorded, providing robot motion tracking. The User Experience & Ergonomics module supplies ergonomic cost and joint state data, supporting assessments of operator experience. The Perception & Autonomy module contributes detection results, enabling environment understanding and autonomous decision-making. Safety data are captured through the Safety & Monitoring module, which logs detected events relevant to safety. Finally, the AI Station Controller manages higher-level orchestration by reporting task, action, and schedule status, ensuring synchronized execution across processes and enabling schedule-driven analytics.

Table 2 summarizes the captured information and provides the mapping between the source which can be either ROS or CDC, Kafka Topics and DAR collection name. AISC automatically publishes to the related topic via a Kafka Connector.

Table 2: Additive Manufacturing Pilot Case - DIM Data Capture- Ros Topics to Kafka Topics mapping

Module	Data Captured	CDC	Ros Topic	Kafka Topic	DAR Collection Name
MPCM	Timestamped Robot Joint and Tool state		/abb_publisher/joint_tool_states	MPCM.joint_tool_states	MPCM.joint_tool_states
UXE	Ergonomic Cost		/erg_cost	UXE.erg_cost	UXE.erg_cost
UXE	Joint States		/human/joint_states	UXE.joint_states	UXE.joint_states
PAM	Detection Results		/body_tracking_data	PAM.body_tracking_data	PAM.body_tracking_data
SAM	Detected Events	SAM DB	–	SAM.safety_event	SAM.safety_event
SAM	Devices	SAM DB	–	SAM.device	SAM.device
AISC	Task, Action and Schedule Status			AISC.runtime_status	AISC.runtime_status

3.1.1.2 Aeronautics Manufacturing Pilot Case

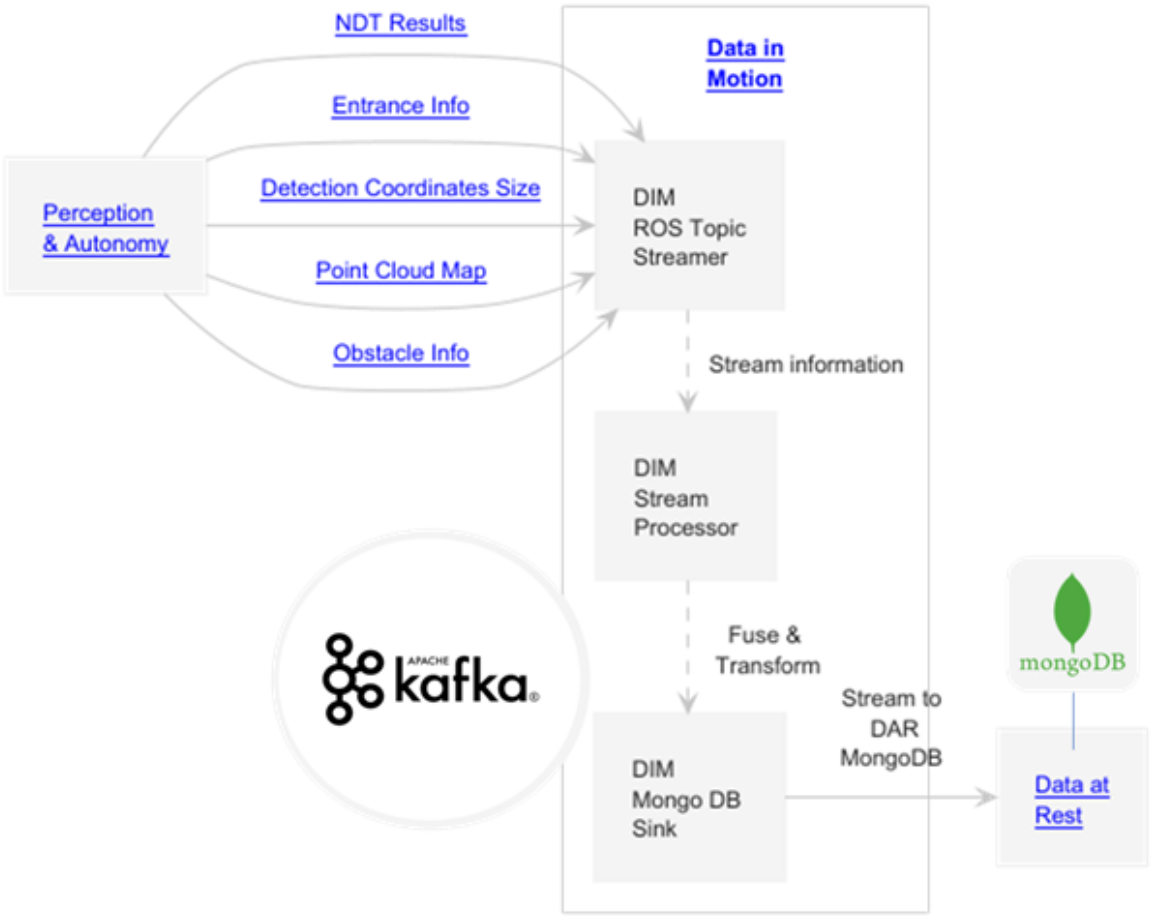

Figure 44: Aeronautics Open Pilot Line – Big Data Pipeline

In the Aeronautics Pilot Case OPL the Big Data Pipeline collects data from PAM. Figure 44 shows a diagram that focuses on the DIM capturing streaming, fusing, transformation and sinking process in the Aeronautics Manufacturing Open Pilot Line. The Perception & Autonomy module captures and processes environment-related data to enable situational awareness and autonomous decision-making. It provides NDT results for localization, entrance information for tracking, and detection coordinates with object size for identification of elements in the workspace. In addition, it generates a map point cloud to represent the environment ensure efficient navigation and collaboration.

Table 3 summarizes the captured information and provides the mapping between the source which can be either ROS, Kafka Topics and DAR collection name.

Table 3: Aeronautics Pilot Case - DIM Data Capture- Ros Topics to Kafka Topics mapping

Module	Data Captured	Ros Topic	Kafka Topic	DAR Collection
PAM	NDT Results	/pam_interfaces/NdtResults	PAM.ntd_results	PAM.ntd_results
PAM	Entrance Info	/pam_interfaces/EntranceInfo	PAM.entrance_info	PAM.entrance_info
PAM	Detection Coordinates & Size	/pam_interfaces/detection_coordinates_size	PAM.detection_coordinates_size	PAM.detection_coordinates_size
PAM	Map Point Cloud	/rtabmap/cloud_map	PAM.cloud_map	PAM.cloud_map
PAM	Obstacle Info	/pam_interfaces/ObstacleInfo	PAM.obstacle_info	PAM.obstacle_info

Figure 38 shows a snapshot of an open source UI for Apache Kafka showing the Kafka topics, number of messages and Kafka Topic message size for a local test deployment, showcasing the corresponding Kafka Topics with the information captured via ROS .

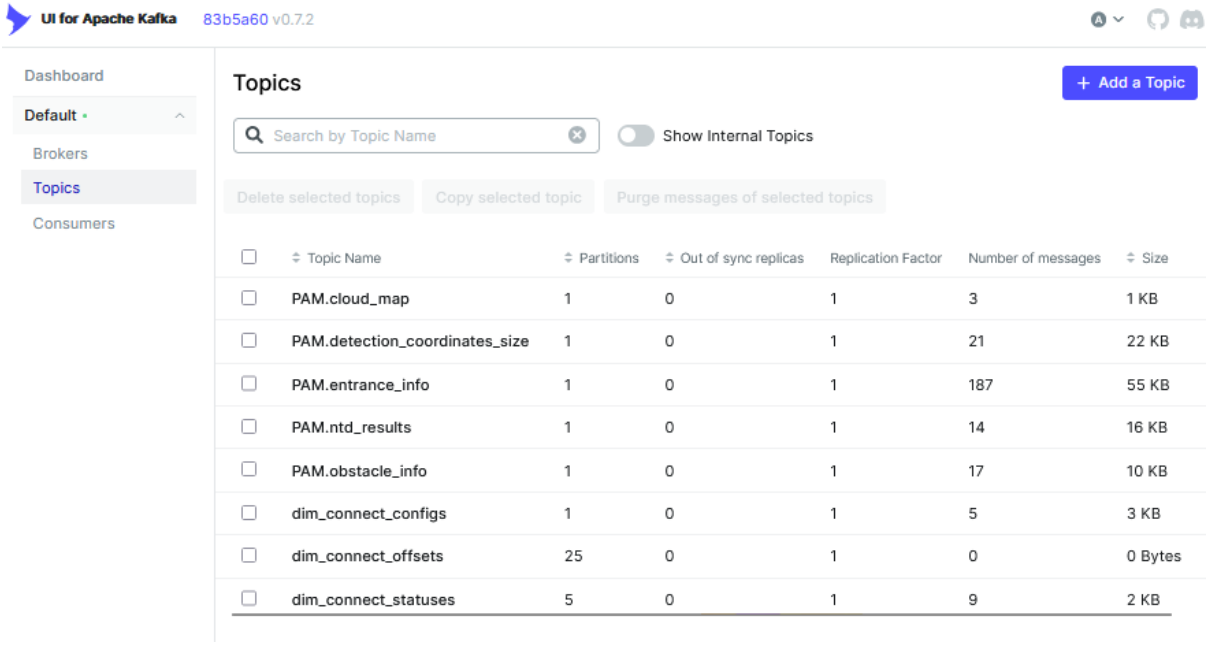

Figure 38: Aeronautics Pilot Case - DIM - Apache Kafka UI Snapshot

3.1.1.3 Automotive Manufacturing Pilot Case

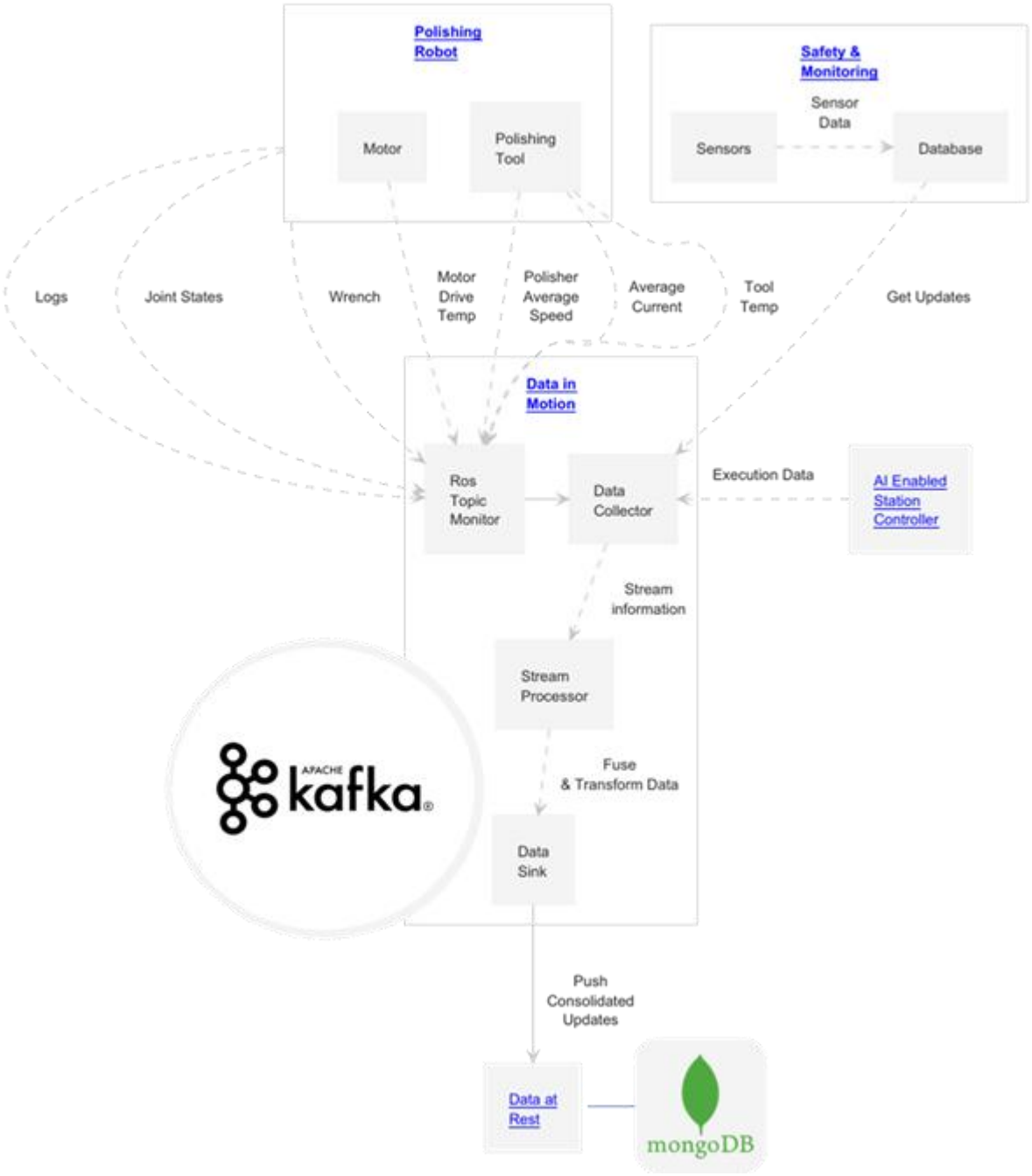

Figure 45: Automotive Pilot Case - Big Data Pipeline M36

In the Automotive Pilot Case OPL the Big Data Pipeline collects data from PAM. Figure 45 shows a diagram that focuses on the DIM capturing streaming, fusing, transformation and sinking process in the Automotive Manufacturing Open Pilot Line. The Data in Motion (DIM) module captures data streams from different sources to acquire real-time information. From the Polishing Robot, DIM collects logs, joint states, wrench forces, motor drive temperature, polisher average speed, average current, and tool temperature, providing detailed insights into robotic operations. From Safety & Monitoring, DIM receives detected safety events, contributing to safety tracking. In addition, from the AI Station Controller, DIM gathers task, action, and schedule status, enabling synchronization and oversight of process execution.

Table 4 summarizes the captured information and provides the mapping between the source which can be either ROS or CDC, Kafka Topics and DAR collection name. AISC directly publishes the status to Kafka via a Kafka Connector.

Table 4: Automotive Manufacturing Pilot Case - DIM Data Capture- Ros Topics to Kafka Topics mapping

Module	Data Captured	CDC	Ros Topic	Kafka Topic	DAR Collection
Polishing Robot	Logs		/rosout	POLISHING_ROBOT.logs	POLISHING_ROBOT.logs
Polishing Robot	Joint States		/ur_controller/joint_states	POLISHING_ROBOT.joint_states	POLISHING_ROBOT.joint_states
Polishing Robot	Wrench		/ur_controller/wrench	POLISHING_ROBOT.wrench	POLISHING_ROBOT.wrench
Polishing Robot	Motor Drive Temp		/mirka_driver/motor_drive_temperature	POLISHING_ROBOT.motor_drive_temperature	POLISHING_ROBOT.motor_drive_temperature
Polishing Robot	Polisher Average Speed		/mirka_driver/average_speed	POLISHING_ROBOT.average_speed	POLISHING_ROBOT.average_speed
Polishing Robot	Average Current		/mirka_driver/average_current	POLISHING_ROBOT.average_current	POLISHING_ROBOT.average_current
POLISHING_ROBOT	Tool Temp		/mirka_driver/tool_temperature	POLISHING_ROBOT.tool_temperature	
SAM	Detected Events	SAM DB		SAM.safety_event	SAM.safety_event
SAM	Devices	SAM DB		SAM.device	SAM.device
AISC	Task, Action and Schedule Status			AISC.runtime_status	AISC.runtime_status

3.1.1.4 Whitegoods Manufacturing Pilot Case

In the Whitegoods Pilot Case OPL the Big Data Pipeline collects data from multiple modules. Figure 46 shows a diagram that focuses on the DIM capturing streaming, fusing, transformation and sinking process in the Whitegoods Manufacturing Open Pilot Line. The Data in Motion (DIM) module captures data from diverse components to enable real-time data integration and monitoring. From Perception & Autonomy, DIM collects perception data supporting environment understanding. From Collaborative Robot Control, DIM receives CRC actions invocations. The AI Station Controller provides task, action, and schedule status, allowing the monitoring of execution in multiple levels. From User Experience & Ergonomics, DIM gathers the ergonomics score to support human-centric operation. The Dynamic Work & Reorganization module contributes task planning data, while the Humanoid Cobot supplies trajectory goals, all of which enhance collaborative and adaptive manufacturing processes.

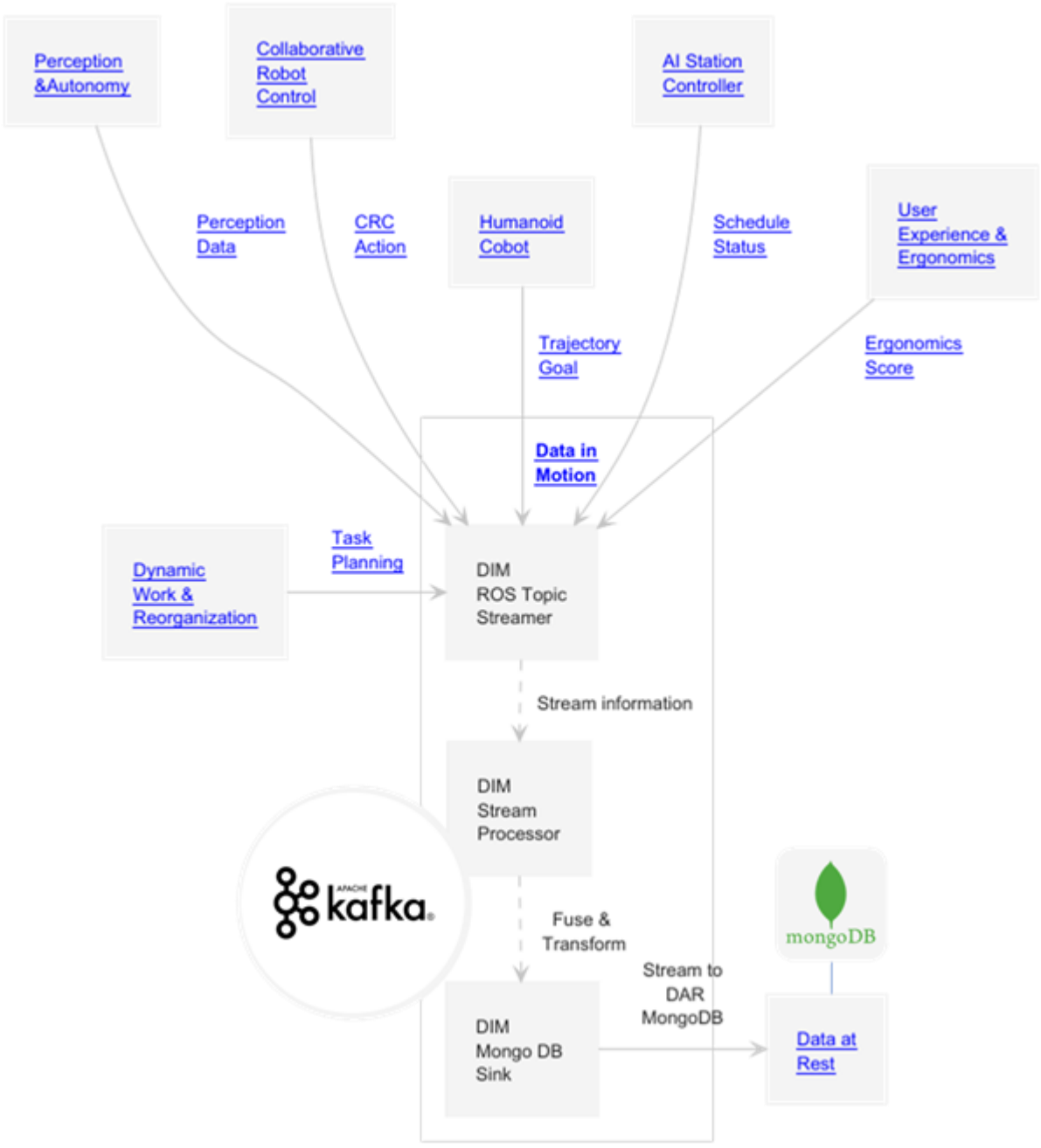

Figure 46: WhiteGoods Pilot Case - Big Data Pipeline M36

Table 5 summarizes the captured information and provides the mapping between the source ROS topics, Kafka Topics and DAR collection name.

Table 5: Whitegoods Manufacturing Pilot Case - DIM Data Capture- Ros Topics to Kafka Topics mapping

Module	Data Captured	Ros Topic	Kafka Topic	DAR Collection
PAM	Perception Data	/tracking_data	PAM.tracking_data	PAM.tracking_data
CRC	CRC Action	/execution/collaborative_robot_	CRC.mpc_action_topics	CRC.mpc_action_topics

Module	Data Captured	Ros Topic	Kafka Topic	DAR Collection
		control/crc_interfaces/actions/back_left/mpc_action/goal		
DWR	Task Planning Data	/execution/integration/task_planner/actions/task_planning/status	DWR.task_planning	DWR.task_planning
AISC	Task, Action and Schedule Status		AISC.runtime_status	AISC.runtime_status
UXE	Ergonomics Score	/ergo_score	UXE.ergo_score	UXE.ergo_score
HUM	Trajectory Goal	/left_arm/cartesian_trajectory_action/status	HUM.cartesian_trajectory_action	HUM.cartesian_trajectory_action

3.1.2 Architecture Implementation Supportive tooling

This section aims to present an overview of the tools and methodology that have been used for the integration and reference software architecture development. The same tools will also be used in WP7 to support the integrated Pilot Case development.

3.1.3 CONVERGING Github Organisation

A GitHub Organization has been created and used for the CONVERGING project with different repositories that are used to persist, share, and track the information needed for the integration and communication architecture development. The contents of the GitHub organization are presented hereafter in Table 6.

Table 6: CONVERGING Organization GitHub Repositories Alphabetical Ordering - M36.

CONVERGING GitHub Organization - Repositories
converging_ros1_interfaces
ConvergingArchitecture
crc_ros1_interfaces
elux_demo_configs
flexbotics_robot_kawada_nextage_v2
hcr-module
hcr-module-interfaces
interfaces
io-drivers
kawada_io_bridge
kawada_nextage_api
kawada_nextage_moveit
kawada_nextage_ros_driver
kawada_ws

CONVERGING GitHub Organization - Repositories

nextage_api

The organization includes 15 repositories that are shown alphabetically in Table 6. Figure 39 presents a screenshot of the repositories via the GitHub webpage. The repositories contents include development related content such as interface definitions, drivers, software, configurations and apis that are useful in terms of integration and configuration.

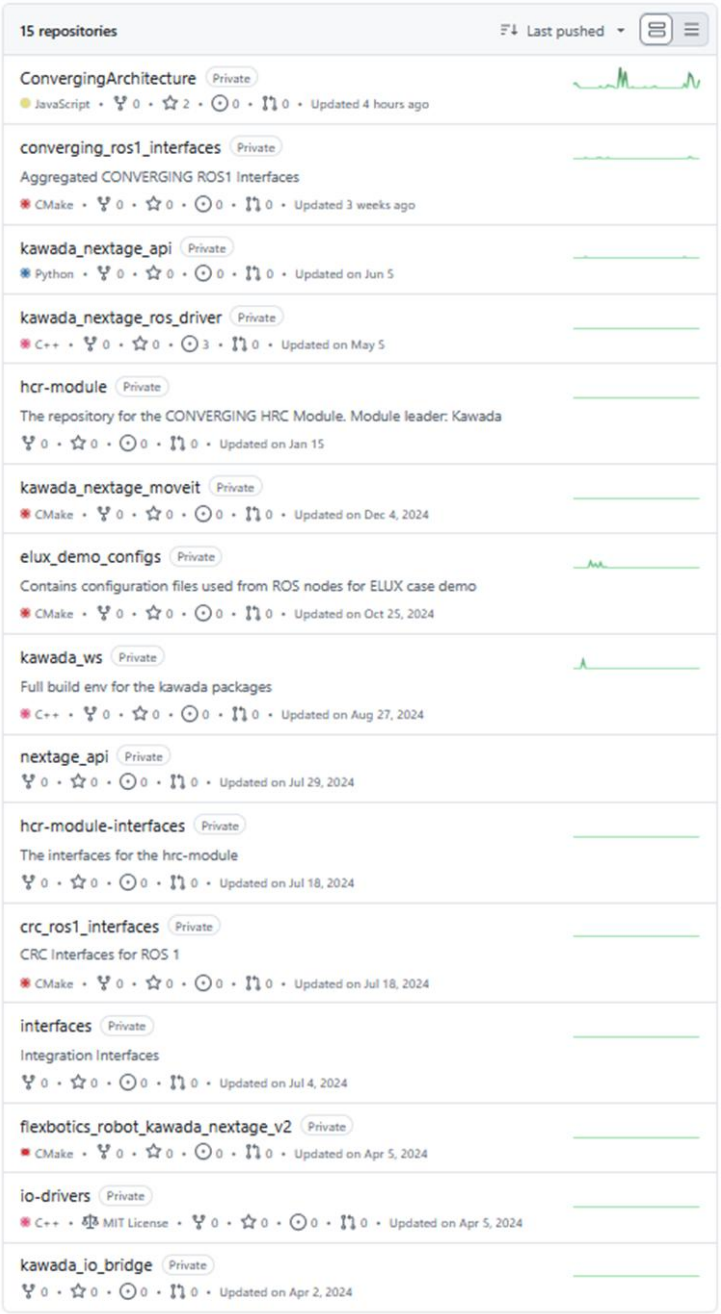

Figure 39: CONVERGING GitHub Repositories - M36.

The use of GitHub to exchange such information is crucial for ensuring transparency, version control, and collaborative development among CONVERGING members. It enables distributed teams to access up-to-date resources, track changes over time, and contribute

improvements or bug fixes in a structured manner. Additionally, by hosting repositories with controlled access, GitHub facilitates both internal coordination. This centralized approach supports reproducibility, accountability, and the efficient integration of system components across different platforms and environments.

The number of registered members in M36 in the CONVERGING GitHub organization is 29. The members are organized in specific, partner-specific teams, and there is also a consortium-wide team. GitHub user teams are created when needed to accommodate appropriate access rights. Figure 40 shows a screenshot of the Teams as shown in the GitHub web user interface through a web browser.

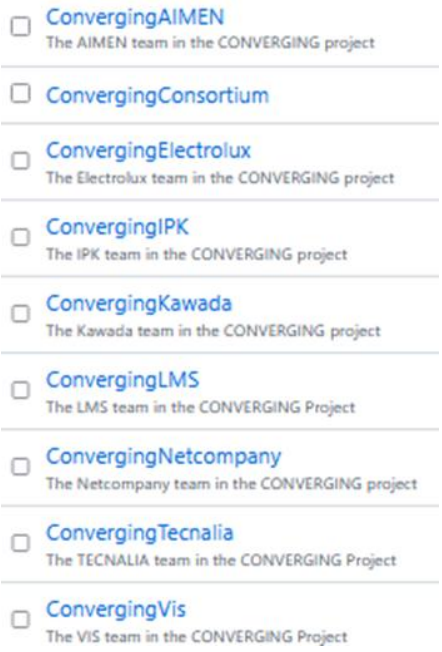

Figure 40: CONVERGING GitHub Teams – M36.

3.1.4 CONVERGING Architecture Repository

A dedicated repository has been used to capture the architecture information of Converging, including interface definitions, diagrams, and tabular organization of information. The repository was called ConvergingArchitecture and can be seen in the list of repositories in Figure 25. The contents of the CONVERGING architecture repository shown in Figure 28.

In particular, the architecture repository includes all diagrams created as part of the CONVERGING Architecture, including module specific and integrated diagrams. These can be further broken down to component and sequence diagrams.

It also includes the interfaces in both ROS1 and ROS2, which are GIT submodules, so they can be used independently for development and a folder with all the docker related configurations for each pilot case organized in separate folders.

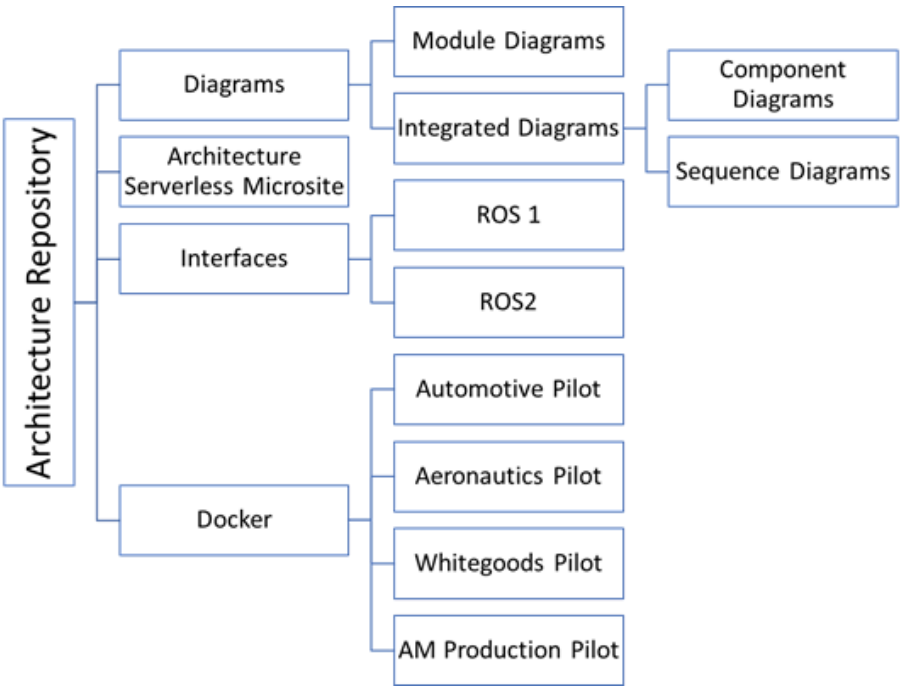

Figure 28: CONVERGING Architecture Repository Contents.

It also contains; a pre-rendered serverless site page has been created that facilitates the browsing of the information without the need of using a web server. A screenshot of this architecture page is shown in Figure 26.

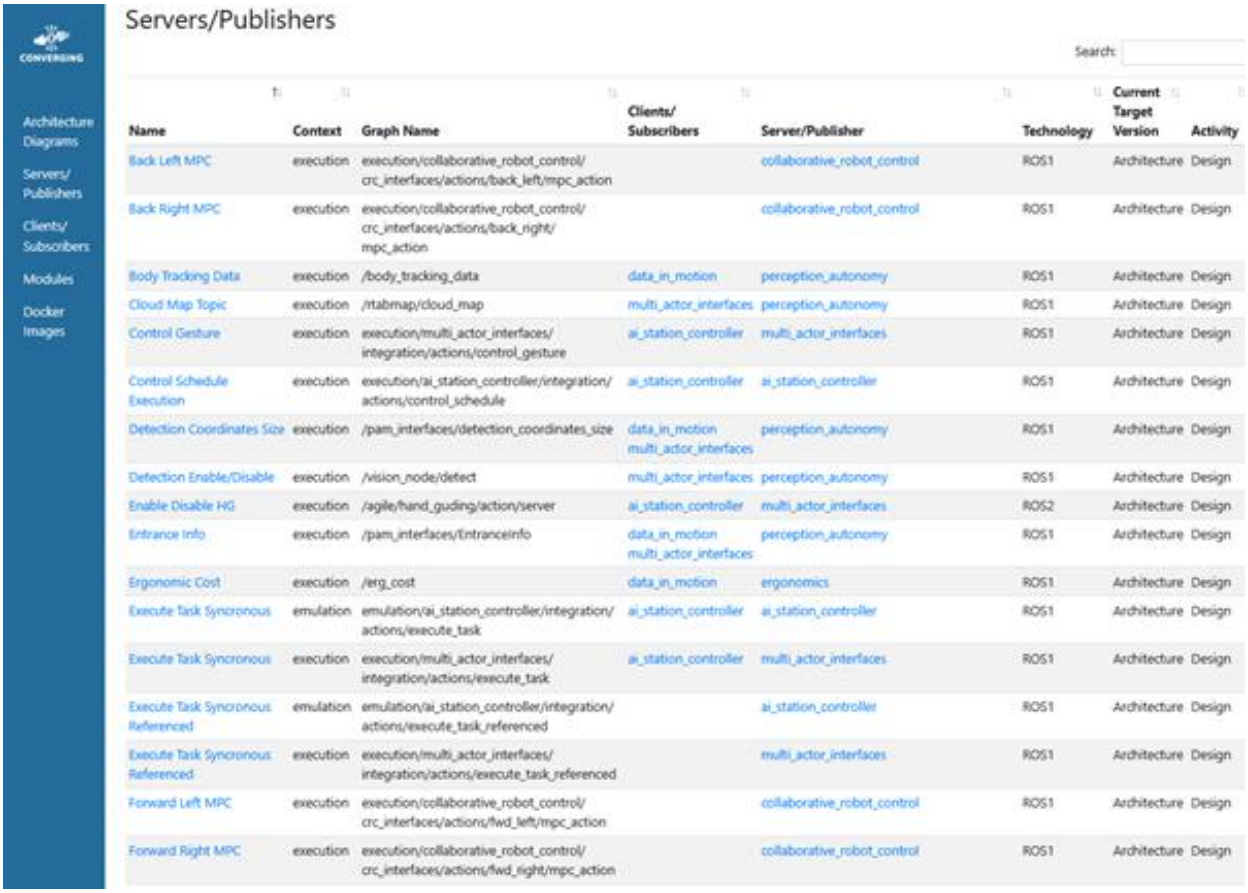

Figure 26: CONVERGING Architecture Server-less Microsite – Servers/Publishers.

The Architecture server-less microsite offers a searchable, tabular and interconnected view of architectural information accessible through any modern browser. The microsite includes the Architecture UML Diagrams, the interface definitions, connected with the modules that consume or provide them and the modules descriptions linked with UML diagrams. The Architecture UML Diagrams include component and UML diagrams. They are uniquely named and identified with a specific id so that they can be referenced easily. In addition, each diagram is connected to the related Pilot Cases. Integrated diagrams offer direct URL links to the interface definitions and module diagrams, for included CONVERGING modules. They are listed in a searchable table that can help the user find a diagram quickly.

The Interfaces Definitions are presented in two tables. A servers and publishers table and clients and subscribers' table. The Servers and Publishers table lists the servers and publishers that are provided by the CONVERGING modules, and all the related interface definitions are captured in a table. The table is searchable and for each interface definition the technology, for instance ROS1 or ROS2, publishing/offering module and graph name are provided.

The clients and subscribers table lists the clients and subscribers expected of the CONVERGING integrated system. The table is searchable and for each interface definition the technology, for instance ROS1 or ROS2, publishing/offering module consuming modules, and graph name are provided.
Both of these tables only list interfaces that are offered for inter-module communication. These interfaces are therefore meaningfully purposed for being accessed from other modules.

The modules table lists all the CONVERGING modules and responsible partners. Each module links to the dedicated module UML diagram.
The Architecture Serverless Microsite offers interactive connections between its different parts that are accessible via browser is shown in Figure 27.

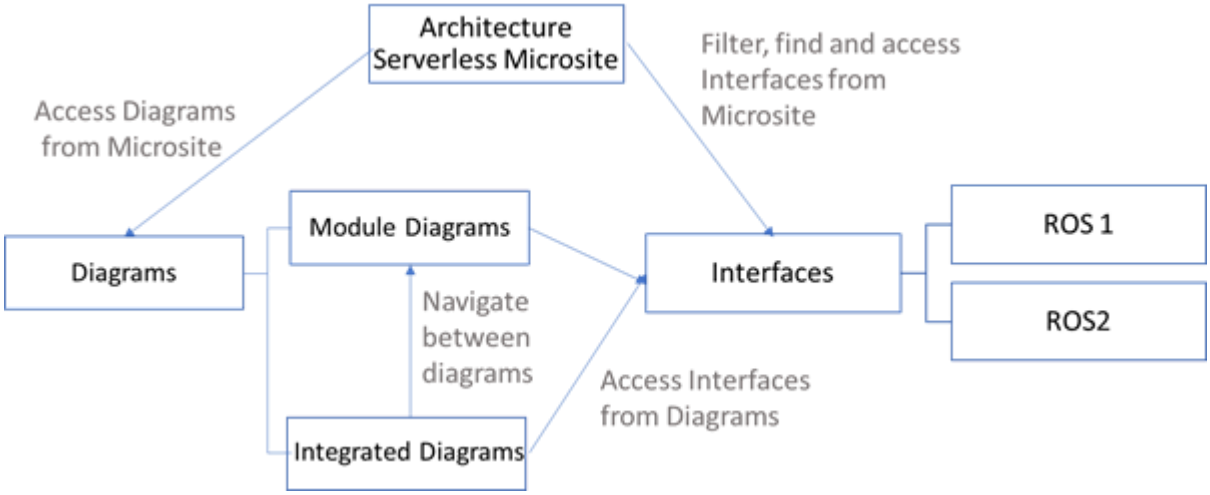

Figure 27: Serverless Microsite - Interlinked Architectural Elements.

Each docker integration folder for the CONVERGING pilot cases includes information in the form of a markdown file named ReadMe.md, a ".env" file containing environmental variables configuration for the used docker images and a set of docker-compose files that are used to pull and run the latest versions of suitable docker images with CONVERGING software modules. This organization, that was used for Big Data Pipeline configuration, especially the DIM and DAR connection is illustrated in Figure 29.

Markdown is a lightweight plain-text markup language for formatting text using simple symbols, for instance # for headings, * for emphasis, and ` for code.

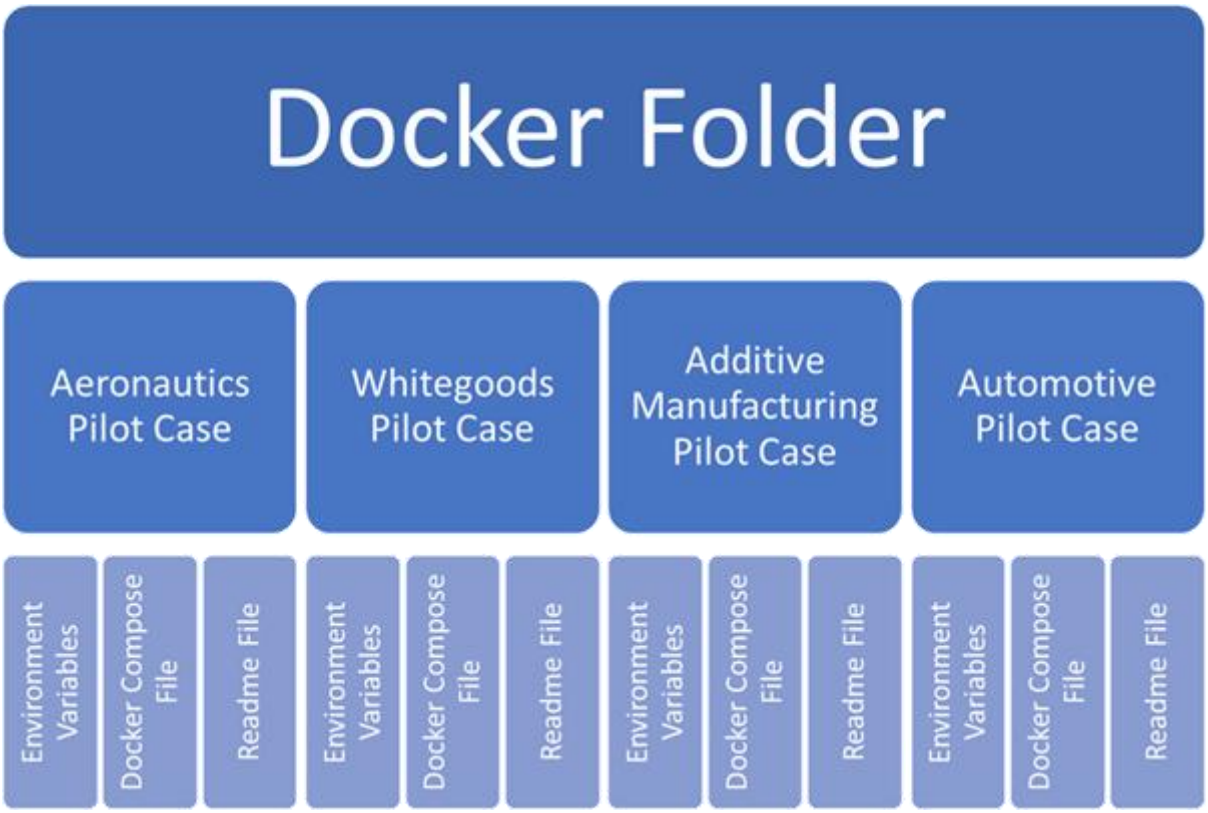

Figure 29: Big Data Pipeline OPL Integration Settings Layout in Git.

3.1.5 CONVERGING Private Docker Repository

Docker is an open-source platform that simplifies the deployment of software by packaging applications and their dependencies into lightweight, portable containers. The Docker containers guarantee a consistent execution environment across heterogeneous computing environments and help reduce integration complexity.

In CONVERGING, a private Docker repository has been established as a central distribution mechanism for software image containers among consortium partners. This Docker repository enables partners to exchange modules efficiently and securely, build integrated testbeds, and deploy multi-container environments using orchestration tools such as docker-compose.

The use of containerization is supported by the CONVERGING Reference Architecture and backbone modules of the integration such as the OpenFlow-based AI Station Controller (AISC) and DIM (Data In Motion). By using Dockerized components, the architecture ensures modularity, reconfigurability, and scalability—allowing the system to adapt seamlessly to diverse pilot cases and evolving industrial requirements. In this way, containerization is not only a technical enabler for deployment but also a key mechanism for maintaining digital continuity and supporting the flexible integration philosophy of the CONVERGING project.

3.1.6 Collaboration and Deployment Observations

During the initial phases of the CONVERGING project, the number of GitHub teams and repositories increased steadily as new partners joined and development activities expanded.

Between M30 and M36, however, these figures have remained stable, reflecting the project's maturity and the consolidation of its collaborative structures. This stability indicates that the core infrastructure, organizational setup, and codebase are now well established, reducing the need for additional teams or repositories. At this stage, development efforts are therefore focused more on refinement, integration, and consolidation rather than on setting up new components.

In parallel, the steady release of updates in the private CONVERGING Docker repository demonstrates the consortium's continued progress in enhancing modules and supporting the deployment of Pilot Cases. These ongoing releases provide evidence of active development, while also reinforcing the stability of the project's collaborative and technical foundations.

This ongoing activity is further confirmed by the progress achieved in the advanced demonstrations within the Open Pilots, where the CONVERGING modules and architecture are actively deployed and validated. Together, these observations underline the effectiveness of the established collaboration model and deployment mechanisms, providing a solid basis for the methodology, organization, and tools described in the following section.

3.2 Next Steps

The CONVERGING integration and communication architecture has been successfully used in the implementation of the CONVERGING integrated system in the CONVERGING OPLs.

The CONVERGING consortium will continue using the established approach in context of WP7, which aim to implement the final versions of the CONVERGING Pilot Cases. Any enhancement or modification might be needed in the CONVERGING Architecture, based on the outcome of the OPLs, will be reported in WP7.

4 Conclusions

This document presents the final findings and developments of Work Package 4, Big Data Pipeline. In particular, the final outcomes of tasks Task 4.3 ” Differentiable digital twins for optimized Machine Learning and AI based planning” and Task 4.4 “Open integration and communication architecture for reconfigurable production – deployment”.

The aforementioned tasks were responsible for the development of the CONVERGING modules.

AI Digital Twin (AIDT)

This deliverable presents the final version of the AI Digital Twin (AIDT) developed within the scope of Work Package 4 (WP4) of the CONVERGING project. AIDT ensures digital continuity across the entire system lifecycle, leveraging the digital thread to accelerate deployment and enhance operational efficiency. Its development has been closely aligned with the requirements of the four pilot cases, integrating virtual representations of shop floors, simulation libraries, planning module interfaces, and communication interfaces with ML/AI frameworks. Throughout this period, the AIDT has matured to support advanced robotic behaviours, including complex kinematics, realistic robot simulation, and trajectory planning. It also incorporates robust communication interfaces and Python 3 support for virtual controllers and libraries. These capabilities have been complemented by the extension of simulation libraries and ergonomics evaluation tools, with continuous integration and fine-tuning carried out in collaboration with pilot partners until the conclusion of the project.

Integration and Communication Architecture

The CONVERGING architecture is modular and reconfigurable and serves as the backbone for connectivity and interaction among production resources, software, and modules. Integration and Communication Architecture include the interfaces and information flow between different software modules as well as a set of supporting tools and methodologies that have been used towards the development of the integrated CONVERGING system in the CONVERGING OPLs.

The progress in WP4 has been closely linked with advancements in other technical work packages (WP3, WP5, and WP6) and the integration efforts in four demonstrators under WP7 and will be used for the integrated development in Pilot Cases that takes place in WP7.