

VIRTUAL COMMISSIONING APPLIED TO ROS AIRCRAFT ROBOTIC CELL

Ariéle Teixeira Ferraz[1], Caroline Cristine Duarte da Silva[2], João Marcos Gomes de Mello[3] & Emilia Villani[4]

[1,2,4]Instituto Tecnológico de Aeronáutica, ITA
[3]EMBRAER

Abstract

Virtual commissioning enables early validation of manufacturing cells, reducing costs and risks associated with the late integration of devices and systems. This paper proposes an incremental virtual commissioning framework for ROS-based control systems applied to aircraft robotic manufacturing cells. The purpose is to overcome some of the limitations that inhibits the use of ROS in aircraft industry. The approach progresses from software-in-the-loop to hardware-in-the-loop configurations, allowing systematic verification of functional and real-time requirements prior to final commissioning. The framework is evaluated through a case study of a velocity-adaptive robotic cell where the material deposition in aircraft joints/gaps is compensated through real-time velocity modulation using a 2D laser beam sensor. Preliminary results demonstrate stable simulation–hardware integration and consistent execution.

Keywords: Aircraft manufacturing, Virtual Commissioning, ROS (Robotic Operating System), Simulation, Verification and Validation

1. Introduction

In the development of manufacturing cells, the commissioning phase refers to the steps of installation, integration, and testing of the cell's control systems [1]. Traditionally, the 'commission' phase occurs late in the development cycle when the infrastructure is in place and equipment is available. However, correcting errors in this phase is expensive and time-consuming [2]. In order to overcome this challenge, the idea of a 'virtual' comissioning was proposed around 2007, inspired by the concepts of 'hardware-in-the-loop' and 'software-in-the-loop' from embedded systems development. Basically, 'virtual commissioning' consists of using digital simulation models of the manufacturing cell to test, validate, and improve a control system before it is physically built. Although the proposal of virtual commissioning is not new, it gained momentum with the introduction of Industry 4.0 technologies and the possibilities that arise from the evolution of the digital model into a digital shadow (the digital model is automatically updated by data from the corresponding physical system, but cannot influence the physical system in return) and a digital twin (the digital model is a dynamic representation that both receives data from and sends data to the corresponding physical system) [3].

In this context, this paper proposes a framework to apply the concept of 'virtual commissioning' to the development of ROS-based control systems for aircraft robotic manufacturing cells. ROS (Robot Operating System) is an open-source middleware framework that provides the software infrastructure to build robot control systems. Among the advantages of ROS are modularity and reusability. The

https://doi.org/10.71945/icas2026_0198

software is organized in independent nodes. Components can be developed, tested, and replaced independently, facilitating reuse. Another advantage is the availability of routines and packages, developed by a large community of ROS users, for different robotic applications such as navigation, SLAM, perception, among others. Finally, ROS provides hardware abstraction and portability to devices of different manufacturers while maintaining vendor independence. Despite these advantages, the adoption of ROS in aircraft manufacturing cells has not yet become a reality. The stricter requirements of aircraft industries, the certification and regulatory constrains, and the need for full traceability and trustworthy verification and validation processes, especially for real-time applications, remain significant challenges that have yet to be overcome.
Following an incremental virtual commissioning approach, this paper explores the use of a virtual simulation environment to test and verify the functional and real-time requirements of robotic cells, first using a software-in-the-loop configuration and subsequently through a hardware-in-the-loop testbench.
This paper is organized as follows. Section 2 introduces the proposed virtual commissioning approach. Then, Section 3 describes the aircraft manufacturing cell used as a case study to evaluate the proposed framework. Section 4 presents preliminary results. Finally, Section 5 draws some conclusions and discusses future work.

2. The Virtual Commissioning Approach

Considering a generic robotic cell composed of a) aircraft manufacturing parts, b) robotic systems (including their controllers), c) end-effectors, sensors, actuators and other controlled devices, and d) one or more control systems, the proposed virtual commissioning approach is organized into 4 steps, as follows:

Step 1 – Model-in-the-loop
In this step, the designed robotic cell is modeled in a robotic simulation environment. This model shall include items a), b) and c) from the above list. The controllers can be either simulated in the same environment or modeled in an external tool, such as MATLAB/Simulink, that must then be integrated with the robotic simulation environment. Examples of simulation environments are Coppelia Sim and Gazebo.

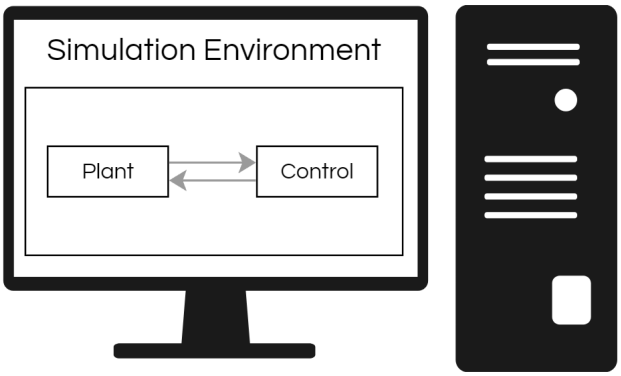
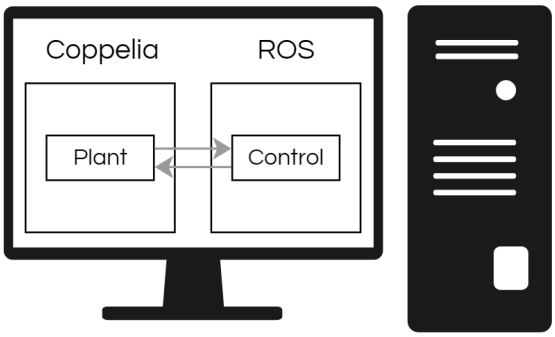


Figure 1 – Model-in-the-Loop (MiL) configuration: the control algorithm and the plant model run entirely within the simulation environment, with no physical hardware involved.

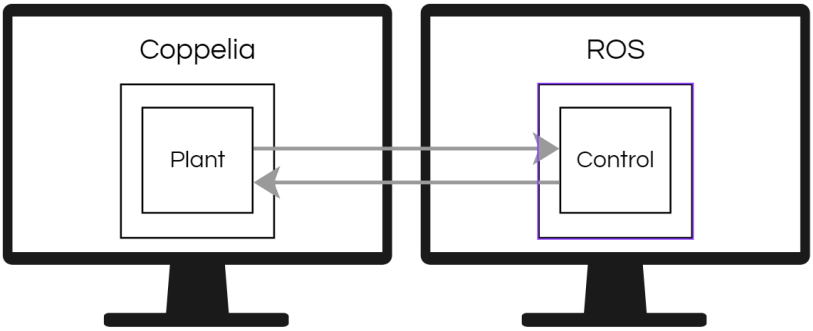
A critical challenge at this stage is model fidelity. In the context of the robotics field, typical issues include incorrect kinematic model parameters of a manipulator robot (e.g., Denavit-Hartenberg parameters, joint limits, and payload definitions), errors in the sensor signal generation pipeline and inaccurate dynamic parameters of a robotic mobile base. Standard kinematic models account only for geometric parameters; non-geometric factors such as joint compliance and backlash, which significantly affect positioning accuracy under tool payload, are typically omitted [4, 5].
The simulation environment must also represent the dynamic behavior of the mobile manipulation platform with sufficient accuracy, including the interaction forces between the mobile robot base and

the floor, the compliance of the robot joints under tool payload, and the optical geometry of the laser line scanner. This simplification tends to produce idealized outputs free of noise, drift, and occlusion, causing the control algorithm to appear more robust than it will be in practice.
Simplified sensor models, in particular, exert a dual and opposing influence on algorithm validation. An under-modeled sensor – one that omits realistic noise sources, drift, and surface-dependent artifacts – yields excessively clean outputs, causing the control algorithm to appear more robust than it will be in practice [6]. Conversely, a sensor model that over-estimates noise intensity or introduces artifacts inconsistent with the actual sensor behavior leads to unnecessarily conservative tuning, masking real deficiencies that would only become apparent when the algorithm operates on data from the physical sensor [7].
Similarly, the control algorithm, which in the final system runs on a dedicated embedded board, is executed on a personal computer during this step. As a consequence, computational constraints such as execution time overruns, memory limitations, and fixed-point arithmetic effects specific to the target hardware are not observable at this stage.

Step 2 – Software-in-the-loop
In this step, the previously verified control routines are implemented in ROS and integrated with the robotic simulation environment. Both the ROS controllers and the robotic simulation run in the same computer, as independent processes.
ROS naturally adopts a modular architecture, where nodes run independent executable processes. Topics and services provide communication mechanisms. The concept of Docker and containers is used to package and run the ROS application with all its dependencies (ROS version, libraries, drivers, configurations). The application itself can be organized in multiple containers, or in one container hosting multiple nodes.
Finally, tests are run to evaluate functional requirements, synchronizations, and coordination of the ROS components. This is the step that validates the integration between the control node, the sensor processing node, and the robotic cell motion execution node.
The main challenges at this stage arise from the transition of the algorithm from a general-purpose execution environment to a ROS-based architecture with explicit communication contracts. Message serialization and deserialization introduce latency in the control loop that was absent in Step 1. As ROS 2 documentation and related studies make explicit, some control systems require not only fast execution but bounded release time and bounded computation time, since variable delay – rather than fixed latency – acts as a time-varying phase lag that erodes the phase margin of the controller and injects noise into derivative terms [8, 9]. High-frequency topics such as laser scanner point clouds are particularly sensitive to queue overflow, which can cause silent data drops affecting the application.
Furthermore, since the algorithm is still executed on the host machine rather than on the final embedded board, architecture-specific behaviors – such as differences in floating-point handling between x86 and ARM processors, or the absence of a real-time operating system – are not yet exposed [10]. Timing issues between the velocity command output and the robot's motion execution, which are critical for the continuous material deposition task, may therefore go undetected until the next step.
Step 3 – Hardware-in-the-loop
In this step, the ROS routines are progressively migrated to their final embedded target while remaining integrated with the robotic simulation environment. Both functional and real-time requirements are evaluated, as well and synchronizations and timing issues. Depending on the complexity of the robotic cell, additional activities may include the gradual replacement of simulated robot, sensor, and actuator models with their corresponding physical devices.
This is the most demanding step of the virtual commissioning process for the present case study, as it directly exercises the constraints of the embedded board that runs the control algorithm. The migration of the ROS nodes running in a personal computer to the embedded target commonly reveals execution time overruns in the control loop, caused by CPU limitations, competing interrupt service routines, and thermal throttling of the embedded processor under sustained load [11].
Communication latency and jitter between the embedded board and the robot controller must be



(a) Control algorithm and simulated plant in the same general-purpose computer.



(b) Control algorithm and simulated plant in different general-purpose computers.

Figure 2 – Software-in-the-Loop (SiL) configuration. This step can run in the same computer 2a or on different computers 2b.

characterized and accounted for in the adaptive control law, as uncompensated delays directly impact the accuracy of material deposition, particularly during gap transitions and intersections.
Integration with the physical robot controller (KUKA Sunrise OS) via its communication interface also introduces constraints not present in the simulated environment, such as command cycle synchronization and safety response times. Compatibility between different robot controller vendors and simulation environments is a recognized challenge in virtual commissioning of robotic cells [12, 13]. Additionally, the real-time laser scanner data stream, when processed on the embedded target, may not sustain the throughput achieved on the host workstation, requiring algorithm optimization or a reduction in scan frequency [6].

Step 4 – Final commissioning
Final commissioning is the last step before a system is released for regular operation. It validates that the fully integrated, real system meets all functional, performance, safety, and regulatory requirements in its actual operational environment.
This step typically surfaces residual discrepancies between the virtual models and the physical system [14]. For the present case study, expected challenges include deviations in the robot's absolute positioning accuracy due to joint compliance under the weight of the dispensing tool and laser scanner – a dynamic effect that simplified kinematic models do not capture [5, 4]. The omnidirectional mobile base may exhibit floor-contact irregularities and wheel slippage not present in the simulation, which can introduce positional drift during horizontal joint operations. Such non-systematic odometry errors, originating from surface unevenness and wheel-contact conditions, are not possible to predict or reproduce in a virtual model [15].
Electromagnetic interference generated by the robot joint drives may also affect the embedded board and sensor cabling, producing spurious readings in the laser scanner output. It has been reported that industrial robots can cause receiver sensitivity losses of up to 13 dB in nearby electronics,

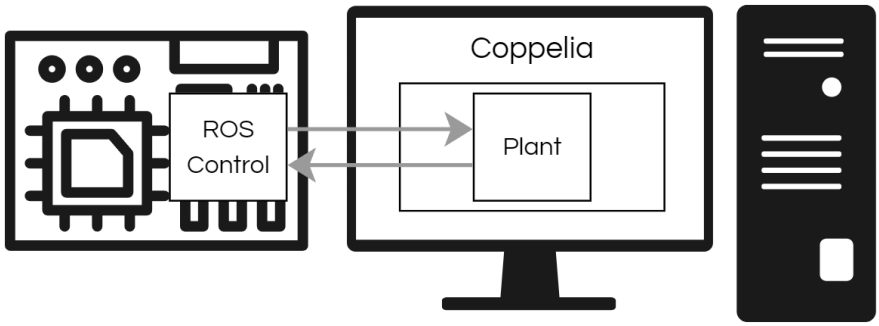


Figure 3 – Hardware-in-the-Loop configuration

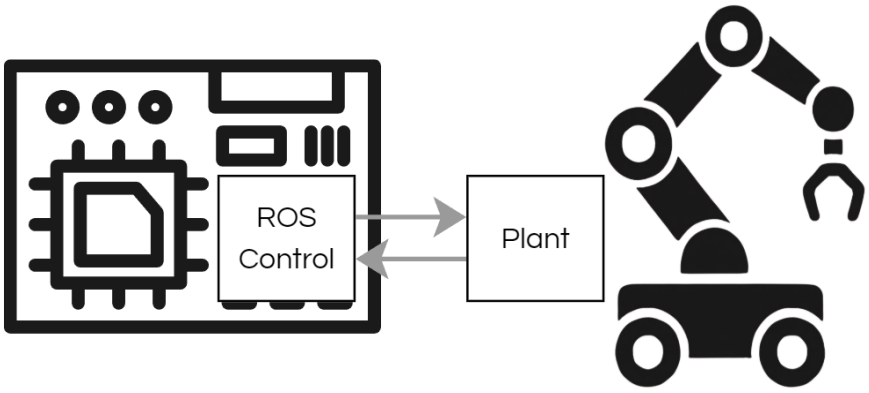


Figure 4 – Final commissioning configuration.

with DC-DC converters and connecting cables acting as unintentional antennas [16]. Finally, real aerostructure panels introduce variability in surface reflectivity and gap geometry that synthetic simulation environments cannot fully anticipate, requiring final calibration and robustness validation of the gap estimation algorithm under operational conditions.
This work focuses on Steps 2 and 3. In the next section, it summarizes those steps for an aircraft manufacturing cell.

3. Case-study: Aircraft Manufacturing Cell

The case study adopted in this work is that of a velocity-adaptive control of a mobile robotic system that performs a continuous operation of depositing material in a gap or joint of aircraft components. Examples of processes that fit into this description are the sealing of joints between aircraft structural components or the welding of non-critical internal parts.
The main challenge is that, due to the variability of the gap/joint between the components or the presence of intersections, the amount of material to be deposited must be controlled in real-time. Despite the importance of these operations, they remain largely manual due to the difficult of dealing with the process variability, which challenges conventional automated systems [17].
Traditional robotic approaches rely on continuous joint geometries and flow-based adaptation, which increase system complexity and reduce flexibility [17, 18]. Recent works have explored vision- and force-based monitoring for adaptive control [19, 20, 21, 22], demonstrating high precision but requiring complex sensing and control hardware.
The solution explored in this work is based on the closed-loop control of the robotic arm (in the case of vertical joints) or the mobile platform (in the case of horizontal joints) to compensate for the process variability. The system uses the output of 2D laser sensor to estimate the gap and detect discontinuities.
The robotic cell adopted as a case study is illustrated in Figure 5.
The experimental setup comprises a robotic manipulator (KUKA LBR iiwa 14) integrated with an

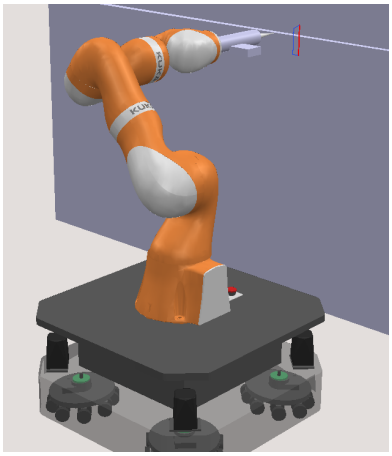


Figure 5 – Simulated mobile manipulator performing velocity-adapted operation over a variable gap in CoppeliaSim.

omnidirectional mobile base, forming a mobile manipulation platform. A simplified material flow dispensing tool, augmented with a laser line scanner, is mounted on the flange as the end-effector. This configuration enables autonomous high-precision gap measurement on the adjacent aerostructure panel.
The manufacturing cell developed in this work integrates real-time gap area measurement to estimate both the available fluid volume and the corresponding flow rate. These parameters govern the binary activation logic of the dispensing end effector, operating under an on/off control scheme. Once material flow is established, the system computes the optimal traversal velocity of the cell, ensuring that the quantity of deposited material precisely matches the volumetric requirements of the gap geometry to fill it adequately, as shown in Figure 6.

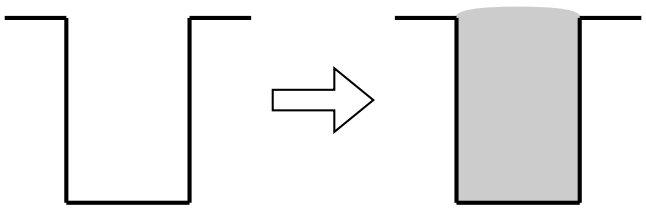


Figure 6 – Empty gap before material deposition to filled gap after velocity-adaptive control pass.

At Figure 7 a fluxogram shows the logic behind the process. The robotic cell is initialized at the origin of the designated filling path. Based on cross-sectional area measurements acquired through onboard sensor data, the cell advances along the trajectory at a velocity inversely proportional to the measured area, ensuring that the deposited material volume remains consistent until the cell completes its pass, even if a variable gap is detected. If a vertical discontinuity is detected along the path, as shown in the Figure 8, the control software records the spatial coordinates of the interruption point. This positional data is subsequently utilized to schedule a return pass, during which the robotic arm performs a corrective filling operation to restore geometric continuity at the identified location.
Compared to existing literature, the originality of this work lies in: (i) using velocity modulation rather than flow or vision-based control for gap adaptation [17, 18, 19, 20, 21, 22]; (ii) applying ROS-based microservices in aerospace manufacturing automation [23]; and (iii) providing a reproducible simulation-driven framework bridging virtual commissioning and embedded systems.

4. Results

This section focuses on the application of Steps 2 and 3 of the virtual commissioning approach to the aircraft robotic cell described in the previous section.
The system is implemented using ROS Noetic, organized as microservices deployed in containers running in Docker [24, 23]. Each functional layer—perception, trajectory planning, velocity adaptation,

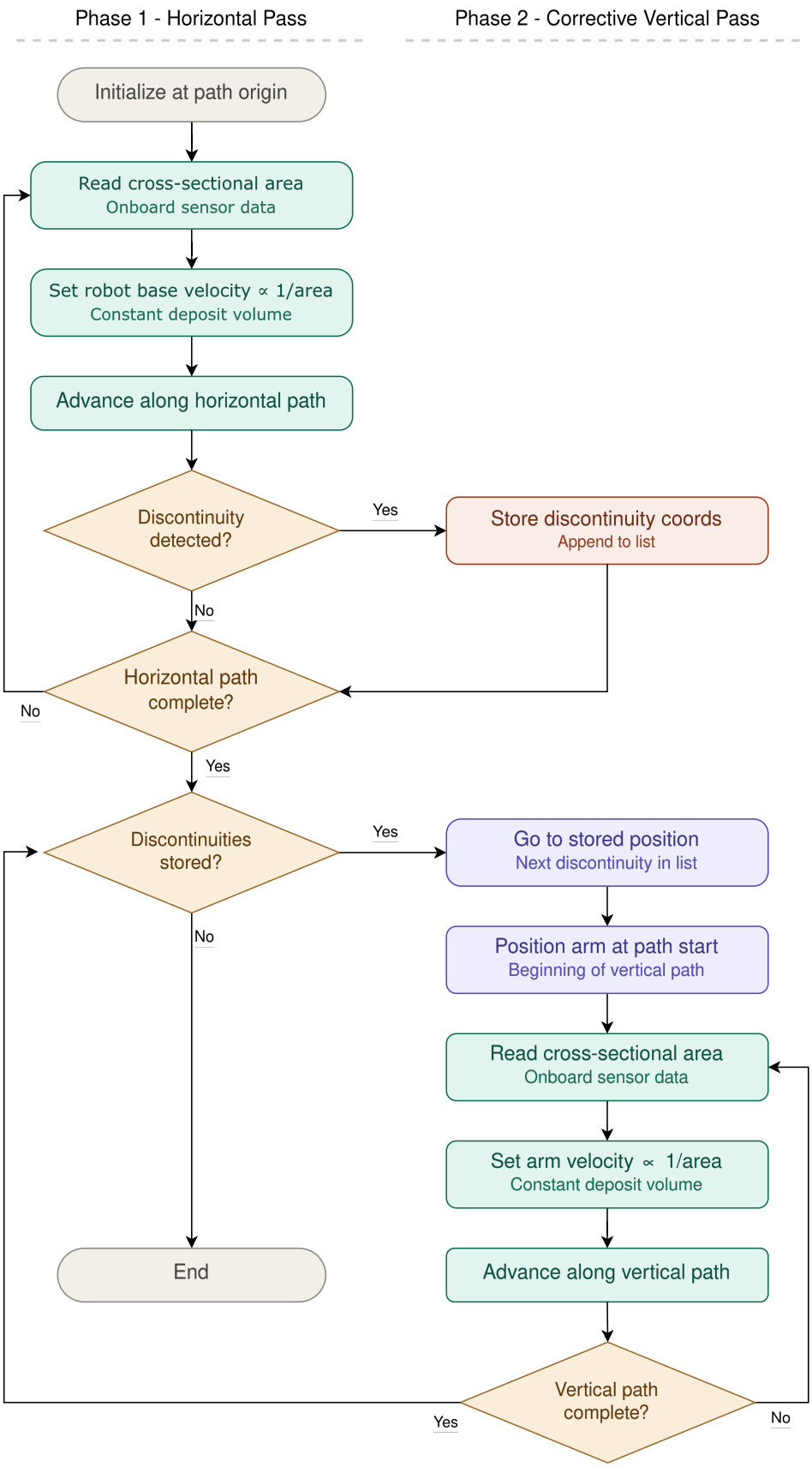


Figure 7 – Flowchart of the robotic cell control logic, illustrating the two-phase filling strategy: the horizontal pass (left) regulating the flow deposition velocity while recording the coordinates of discontinuities; the corrective vertical pass (right) that iterates over all stored locations, performing targeted filling operations to restore geometric continuity.

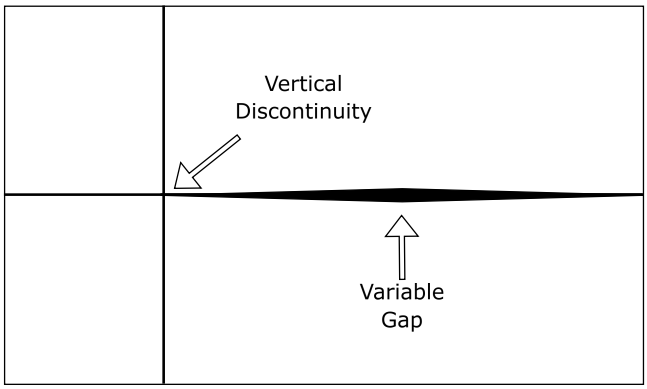


Figure 8 – Detail of a vertical discontinuity and gap variation detected by the 2D laser sensor during the horizontal pass.

and actuation—is encapsulated in independent ROS nodes to ensure modularity and scalability.
As described at the end of the section 2., this case study focuses on two steps of the virtual commissioning approach: Software-in-the-loop and Hardware-in-the-loop. The development of each step is described in the following subsections.

4.1 Software-in-the-loop

After the robotic cell is modeled in CoppeliaSim, the system is connected to ROS so the control routines can be integrated with the simulation environment. For a robotic system, this means developing the application codes and then simulating the system with a simulation environment software – in this case, CoppeliaSim. The ROS framework facilitates the implementation and integration of the application developed with the hardware systems of these models for simulation and tests.
A complete virtual production environment has been developed in CoppeliaSim [25], as shown in Figure 10, including parametrized gap profiles and a vertical discontinuity, a simulated manipulator, and an omnidirectional base. CoppeliaSim supports high-fidelity simulation of robot kinematics, dynamics, and sensor feedback, suitable for digital twin validation [25].
Bidirectional real-time communication between ROS and the simulator has been verified, enabling deterministic execution of velocity-adaptive trajectories.
The adaptive control law is described in Equation 1, where Q is the constant volumetric flow of material and A is the area of the gap/joint, estimated from the width of the gap $gap(t)$ obtained from 2D laser beam sensor.

$$v(t) = \frac{Q}{A(gap(t))} \tag{1}$$

The whole system runs in a general-purpose computer equipped with a 12th Generation Intel® Core™ i5-12450HX processor, 16 GB of DDR5-4800 MT/s SODIMM memory, and an NVIDIA® GeForce RTX™ 3050 6 GB GDDR6 discrete GPU. The operating system used was Ubuntu 20.04 LTS (Linux).
To verify the speed control of the robotic cell with respect to gap variation, the following scenario was constructed: the trajectory begins with a constant gap of 3 mm over 1 m, followed by a vertical step discontinuity, introduced to assess the system response under such abrupt change, as illustrated in Figure 11. After the interruption, the gap increases linearly to 6 mm over the subsequent 1 m, then decreases back to 3 mm over the final 1 m, yielding a total trajectory length of 3 m. The measured gap profile and the corresponding robot velocity variation are presented in Figure 12.
The vertical discontinuity was deliberately introduced at $y = 1.245$ m, and its detection by the system during execution was recorded at $y = 1.257$ m, representing a positional error of 12 mm, which is deemed acceptable for the proposed case study. Upon discontinuity removal, the gap measurements resume smoothly, and the corresponding modulation of the robot velocity can be clearly observed, as presented in Figure 12.

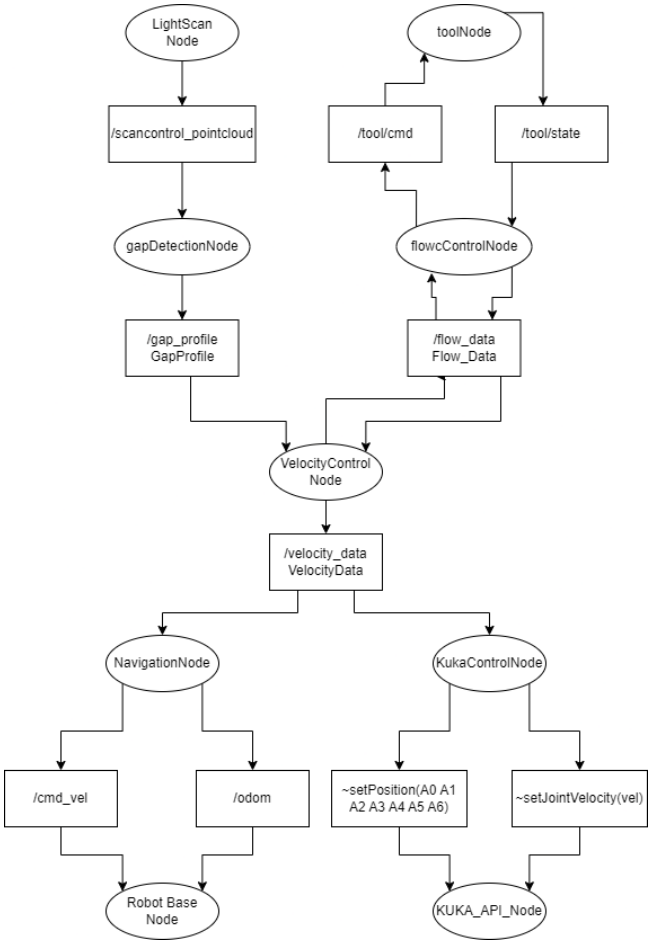


Figure 9 – ROS-based microservice architecture deployed in Docker containers. Each functional layer (perception, trajectory planning, velocity adaptation, actuation) runs as an independent node, communicating via ROS topics.

Although the gap remains constant throughout the first meter, a considerable variation is observed in the measurements acquired by the sensor modeled in CoppeliaSim, yielding a mean value of $\bar{g} = 0.0049 \pm 0.0006$ m, with a minimum of 0.0042 m and a maximum of 0.0052 m. This variation is attributed to the simplified sensor model adopted in the simulation: the physical sensor is a 2D laser with 640 points and a resolution of 19 nm, which was modeled as a 640×1 depth camera, since implementing a full 640-laser model would impose an prohibitive computational load on the simulation environment, rendering the tests infeasible. This simplification precludes a fully validated Software-in-the-Loop (SiL) assessment of the actual sensor noise characteristics. Nevertheless, despite the discrepancy between the modeled and the physical sensor, it can be observed from Figure 12 that the robot effectively tracks the gap variation by modulating its velocity, as evidenced by the close correspondence between the velocity profile and the gap variation profile.

4.2 Hardware-in-the-loop

In the context of the virtual commissioning framework presented in this work, two distinct HiL configurations were investigated. In the first, the control stack was embedded on a Toradex computing module and connected to the simulated plant environment, enabling the validation of the embedded hardware and its real-time software behavior before full system integration. In the second configuration, the ROS-based control algorithm was deployed on a general-purpose computer and interfaced with the real robotic arm executing the target task, allowing the actual plant dynamics to replace the simulated model. Together, these configurations allowed a progressive and structured transition from simulation to reality, consistent with established virtual commissioning practices.

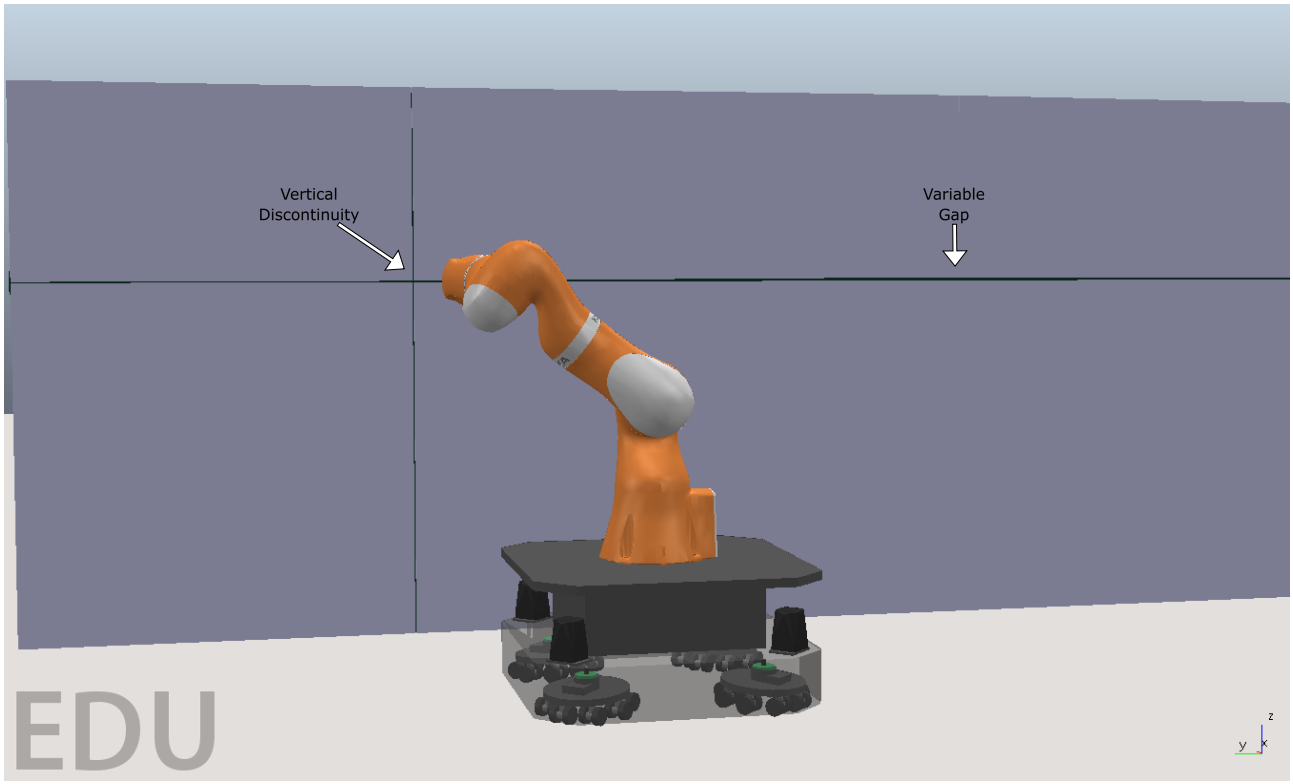


Figure 10 – Complete virtual production environment in CoppeliaSim, showing the mobile manipulator, the parametrised gap profile (including a vertical step discontinuity), and the simulated aerostructure panel.

4.2.1 Controller HiL

This is the classic step for the Hardware-in-the-Loop. This first configuration, referred to as Controller HiL, focuses on the validation of the embedded control hardware. In this setup, the ROS-based control stack developed in the SiL was deployed on a Toradex computing module and interfaced with the simulated plant in CoppeliaSim. The system is shown in Figure 13

This arrangement preserves the virtual environment of the Software-in-the-Loop stage while introducing the actual computational constraints, communication interfaces, and real-time execution behavior of the embedded hardware. The primary objective of this configuration is to verify that the controller performs as intended when running on its final hardware target, before any interaction with the physical plant.

This step employs the same scenario presented in the Software-in-the-Loop phase, shown in Figure 10, as well as a new scenary including only a vertical path, shown in Figure 14. The primary objective of this test is to evaluate whether the system is capable of executing on the target embedded device, in this case a Toradex Dahlia V1.1D. The processor of this board is considerably more constrained compared to the general-purpose computer used during application development and the Software-in-the-Loop phase; therefore, a significant increase in algorithm processing time is expected. However, this degradation is not anticipated to be sufficient to compromise the overall execution performance.

Two scenarios were simulated to compare the computational performance between the two devices: the velocity control of the mobile robotic base for horizontal filling, and the vertical motion of the robotic arm, which, due to the requirement of solving the manipulator inverse kinematics, entails a significantly higher computational load. The data acquired from both tests are summarized in Table 1. The processing times recorded on the general-purpose computer for the horizontal and vertical motions were 4.32×10^{-5} s and X s, respectively, whereas the embedded system required 3.70×10^{-4} s and Z s for the horizontal and vertical motions, respectively. A significant increase in execution time is observed for the Toradex embedded system relative to the general-purpose computer, reaching approximately one order of magnitude, with the embedded system being roughly 8 to 9 times slower.

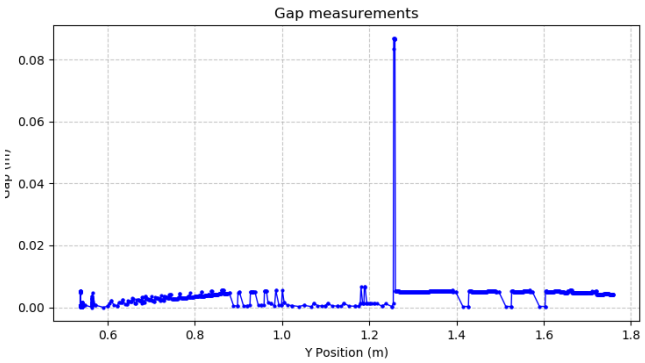


Figure 11 – Gap measurement for the test scenario for the velocity-adaptive control including a vertical step discontinuity measurement.

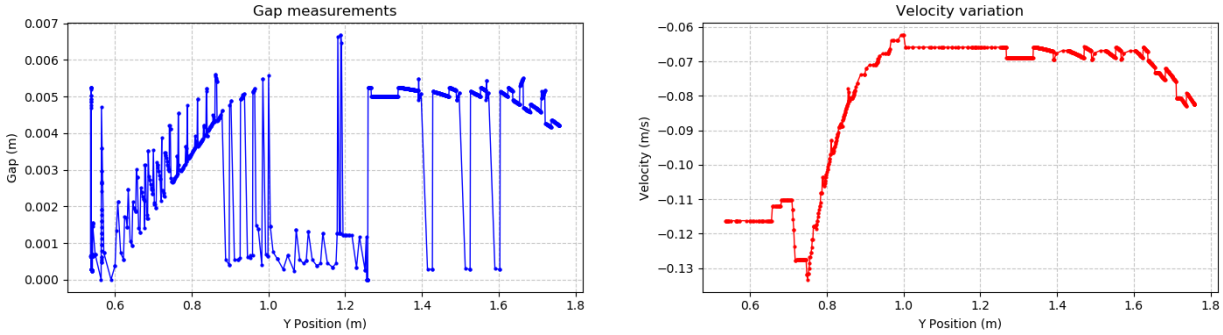


Figure 12 – Measured gap profile (blue) and corresponding robot velocity (red) along the trajectory. Velocity modulation follows the gap variation.

Nevertheless, the execution times for the first scenario remain sufficiently small so as not to compromise the intended operation of the system, and are therefore considered satisfactory for the proposed application. The second scenario involves computationally intensive matrix operations, arising from the continuous requirement of solving the manipulator inverse kinematics, which places a considerably higher processing demand on the embedded board.

4.2.2 Plant HiL

For this second configuration, referred to as Plant Hardware-in-the-Loop, inverts the classic HiL relationship by replacing the simulated plant with the real robotic arm while the control algorithm continues to run on a general-purpose computer. This setup exposes the controller to the actual dynamics, non-linearities, and disturbances inherent to the physical system - effects that are inevitably approximated or neglected in simulation. The Plant HiL stage therefore, serves as a critical intermediate step between virtual commissioning and full system deployment, providing empirical evidence of the controller's robustness under real operating conditions.

In this step, the fixed-base robotic manipulator used was a KUKA LBR IIWA 14, shown in Figure 15, and the control runs in a general-purpose computer equipped with a 12th Generation Intel® Core™ i5-12450HX processor, 16 GB of DDR5-4800 MT/s SODIMM memory, and an NVIDIA® GeForce RTX™ 3050 6 GB GDDR6 discrete GPU. The operating system used was Ubuntu 20.04 LTS (Linux). Similarly to the Hardware-in-the-Loop (HiL) test conducted for the controller, the plant HiL is also evaluated to assess the feasibility of deploying the application on a real robot, examining how the system handles variations introduced by physical sensors and actuators. In addition to the hardware itself, the KUKA setup requires interaction between the software and the robot's proprietary controller, as well as its integration with the API responsible for bridging the KUKA LBR iiwa with ROS, enabling its control. The API employed in this work is an adaptation of [26], freely available on GitHub.

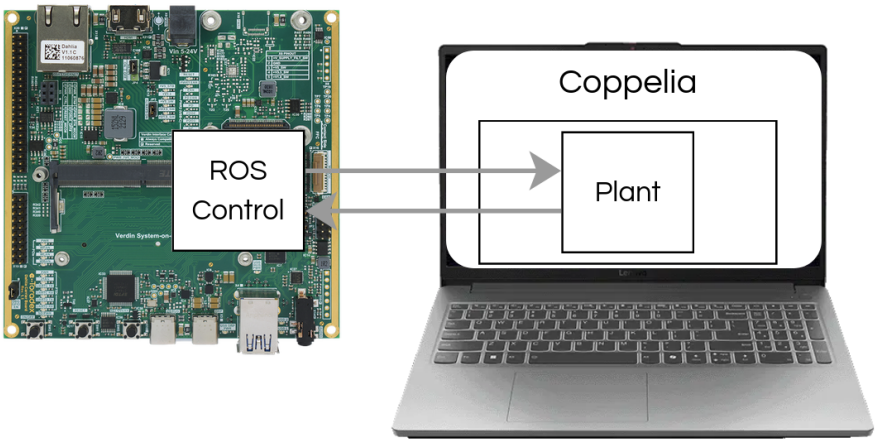


Figure 13 – Controller Hardware-in-the-Loop (HiL) configuration: ROS-based control stack deployed on the Toradex embedded computing module, interfaced with the simulated plant (CoppeliaSim). No physical plant is used.

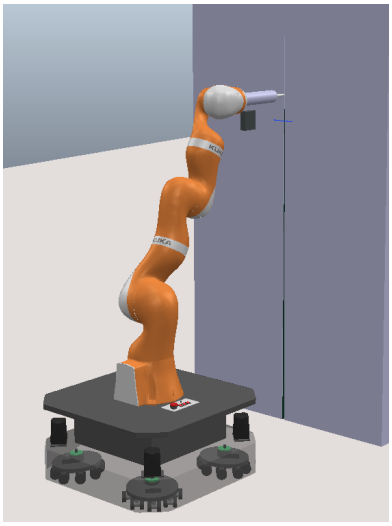


Figure 14 – Vertical-only test scenario for the robotic arm (manipulator) used in the Controller HiL evaluation. The arm follows a vertical path to fill pre-recorded discontinuity points.

In this test, the elapsed time from the moment a motion command is issued to the manipulator until the completion of its execution is evaluated. For comparison purposes, the same measurements were performed in the CoppeliaSim simulation environment through a digital twin, as shown in Figure 16, both with the same velocity, enabling a direct comparison between the physical and simulated systems. Five independent runs were executed, each completing the proposed horizontal trajectory, and the mean and standard deviation of the results are reported in Table 2.
The trajectory was discretized into 100 points, with the motion planning strategy consisting solely of point-to-point displacement. As a consequence of this choice, both the physical IIWA and the simulated arm were unable to execute a strictly straight-line motion in certain runs, requiring intermediate repositioning maneuvers between consecutive points depending on the initial joint configuration — the time associated with these repositioning motions was excluded from the execution time calculations.
A significant difference is observed between the execution times of the physical and simulated systems, and unlike the controller HiL test, this increase in execution time does impact the overall system performance. Such discrepancy is likely attributed to the simplicity of the API employed, which proves insufficient for an application that must operate in real time, given the requirement of continuously dispensing a fluid onto a surface. Furthermore, the need to adapt the simulated arm control to operate under velocity control also contributes to this discrepancy, since the manipulator model effec-

Table 1 – Execution time per control cycle (mean ± standard deviation) for the horizontal (mobile base) and vertical (manipulator) motions, comparing the general-purpose computer (Intel i5-12450HX) and the Toradex embedded module.

	Computer	**Toradex**
Robotic Base	$4.32 \times 10^{-5} \pm 9.51 \times 10^{-6}$	$3.70 \times 10^{-5} \pm 8.43 \times 10^{-5}$
Robotic Arm	$0.1002 \pm 2.3463 \times 10^{-5}$	0.956 ± 0.003

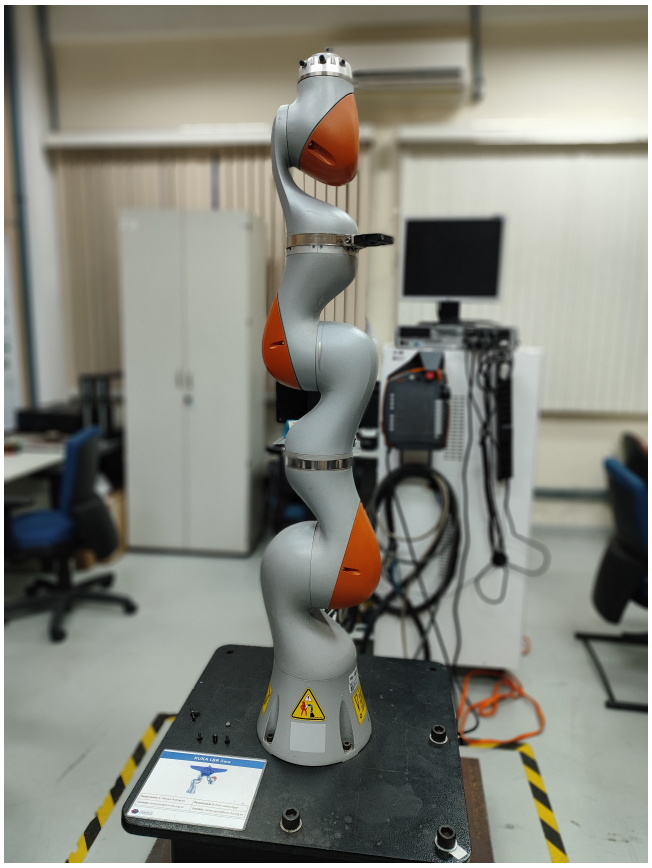


Figure 15 – Plant Hardware-in-the-Loop setup: the real KUKA LBR iiwa 14 robotic arm (fixed base) is used as the physical plant, while the ROS control stack runs on the general-purpose computer. The simulated environment is bypassed.

tively used during software development diverges from the physical system behavior — highlighting the critical importance of adopting the most faithful possible plant model during the commissioning phase.

5. Conclusions

This paper introduced an incremental virtual commissioning framework tailored to ROS-based control systems for aircraft robotic manufacturing cells. The framework progresses from Software-in-the-Loop (SiL) to Hardware-in-the-Loop (HiL) configurations, enabling the systematic verification of functional and real-time requirements before final physical commissioning. The approach was evaluated on a velocity-adaptive robotic cell that performs continuous material deposition in aircraft joints and gaps, using a 2D laser sensor to modulate the robot speed based on the measured gap area.

The SiL stage confirmed that the ROS-based control logic, integrated with the CoppeliaSim environment, correctly implements the velocity modulation law $v(t) = Q/A(gap(t))$. The system reliably detected a vertical discontinuity with a positional error of only 12 mm, which is acceptable for the target application. However, the SiL results also revealed a known limitation: the simplified sensor model (a depth-camera proxy for a 640-point laser scanner) introduces artificial noise and does not fully replicate the behaviour of the physical sensor, reinforcing the need to progressively introduce

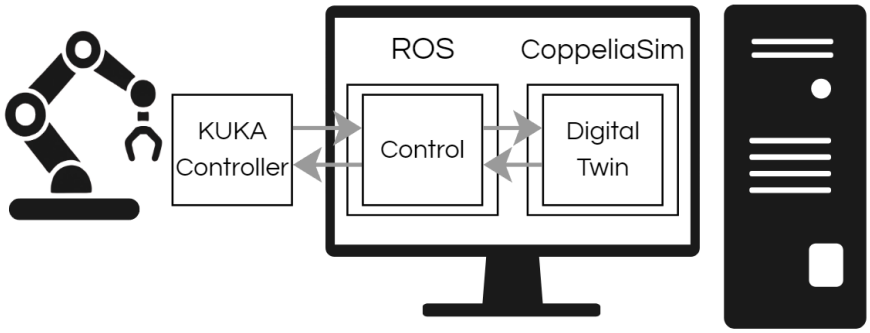


Figure 16 – Schematic representation of the Plant HiL configuration. The control algorithm executes on a general-purpose computer, communicating with the real KUKA manipulator via the API.

Table 2 – Comparison of trajectory execution time (mean ± standard deviation) between the simulated arm (CoppeliaSim digital twin) and the physical KUKA LBR iiwa 14 for the same horizontal path (100 point-to-point displacements, five independent runs).

CoppeliaSim	LBR IIWA 14
0.1502 ± 0.0524	0.7709 ± 0.1206

real hardware.
The HiL stage was divided into two complementary configurations. In Controller HiL, the ROS stack was deployed on a Toradex embedded module while still connected to the simulated plant. This showed that the embedded processor is approximately eight times slower than the general-purpose development computer. Nevertheless, the measured cycle times (0.00037 s for the mobile base and approximately 0.96 s for the manipulator) remain well within process requirements for the mobile base,
confirming that the embedded hardware is adequate for this case, but the robot arm presented doesn't meet those requirements due to the complex math required in the process, as well as waiting for the arm to reach the point to perform this calculation. In Plant HiL, the real KUKA LBR iiwa 14 replaced the simulated plant. This exposed a substantial discrepancy in execution time: the physical arm took 0.77 s to complete the same trajectory that required only 0.15 s in simulation. This difference is primarily attributed to the simplicity of the communication API and the lack of native velocity-mode support in the simulated model, which forced point-to-point displacements. This result reinforces a key lesson of virtual commissioning: the fidelity of the plant model and the emulation of the real control interface are critical. An overly optimistic simulation can hide integration delays and real-time bottlenecks that only become visible when the controller interacts with physical hardware.
Future work will address the remaining stages of the framework. The Final Commissioning stage will integrate the full mobile manipulation platform — including the omnidirectional base and the dispensing end-effector with laser line scanner — operating on real aerostructure panels. Key open challenges include characterising non-systematic odometry errors arising from floor irregularities [15], mitigating electromagnetic interference from robot joint drives on sensor cabling [16], and validating the gap estimation algorithm under the surface reflectivity variability of actual aerospace materials. In parallel, the communication interface with the KUKA Sunrise OS will be replaced by a lower-latency solution to close the execution time gap identified in the Plant HiL experiments.

Acknowledgements

The authors acknowledge the financial support of the following Brazilian Funding Agencies: FAPESP with grants 2021/11258-5, 2025/01408-0 and 2024/00826-0; FINEP with grant 01.22.0313.00.

Copyright Statement

The authors confirm that they, and/or their company or organization, hold copyright on all of the original material included in this paper. The authors also confirm that they have obtained permission,

from the copyright holder of any third party material included in this paper, to publish it as part of their paper. The authors confirm that they give permission, or have obtained permission from the copyright holder of this paper, for the publication and distribution of this paper as part of the ICAS proceedings or as individual off-prints from the proceedings.

References

[1] Peter Hoffmann, Reimar Schumann, Talal Maksoud, and Giuliano Premier. Virtual commissioning of manufacturing systems - a review and new approaches for simplification. pages 175–181, 06 2010.

[2] Gunther Reinhart and Georg Wünsch. Economic application of virtual commissioning to mechatronic production systems. *Production Engineering*, 1(4):371–379, December 2007. Date: 2007/12/01.

[3] Thomas Bergs, Sascha Gierlings, Thomas Auerbach, Andreas Klink, Daniel Schraknepper, and Thorsten Augspurger. The concept of digital twin and digital shadow in manufacturing. *Procedia CIRP*, 101:81–84, 2021. 9th CIRP Conference on High Performance Cutting.

[4] Yuxiang Wang, Zhangwei Chen, Hongfei Zu, Xiang Zhang, Chentao Mao, and Zhirong Wang. Improvement of heavy load robot positioning accuracy by combining a model-based identification for geometric parameters and an optimized neural network for the compensation of nongeometric errors. *Complexity*, 2020(1):5896813, 2020.

[5] Fan Ye, Guangpeng Jia, Yukun Wang, Xiaobo Chen, and Juntong Xi. Kinematic and joint compliance modeling method to improve position accuracy of a robotic vision system. *Sensors*, 24(8), 2024.

[6] Rok Belšak, Janez Gotlih, and Timi Karner. Simulating and verifying a 2d/3d laser line sensor measurement algorithm on cad models and real objects. *Sensors*, 24(22), 2024.

[7] Gonçalo Inácio et al. Quantifying the Sim2real Gap for GPS and IMU Sensors. *arXiv preprint arXiv:2403.11000*, 2024.

[8] Tobias Kronauer, Joshwa Pohlmann, Maximilian Matthé, Till Smejkal, and Gerhard Fettweis. Latency analysis of ros2 multi-node systems. In *2021 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, pages 1–7, 2021.

[9] Sinan Barut, Marco Boneberger, Pouya Mohammadi, and Jochen J. Steil. Benchmarking real-time capabilities of ros 2 and orocos for robotics applications. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 708–714, 2021.

[10] Yuya Maruyama, Shinpei Kato, and Takuya Azumi. Exploring the performance of ros2. 2016.

[11] Yanlei Ye, Zhenguo Nie, Xinjun Liu, Fugui Xie, Zihao Li, and Peng Li. ROS2 real-time performance optimization and evaluation. *Chinese Journal of Mechanical Engineering*, 36(1):144, dec 2023.

[12] Alexandru Andrei, Florin-Adrian Nicolescu, Cristina Pupăză, Cezara-Georgia Coman, and Ionuț Gabriel Ghionea. Applicability of virtual commissioning concepts in industrial robotics as a solution for compatibility issues between virtual simulation and logic control software. *Applied Sciences*, 15(4), 2025.

[13] Roberto Raffaeli, Pietro Bilancia, Federico Neri, Margherita Peruzzini, and Marcello Pellicciari. Engineering method and tool for the complete virtual commissioning of robotic cells. *Applied Sciences*, 12(6), 2022.

[14] Igor Azkarate Fernández, Mikel Ayani Eguía, and Luka Eciolaza Echeverría. Virtual commissioning of a robotic cell: an educational case study. pages 820–825, 2019.

[15] Jordi Palacín, Elena Rubies, and Eduard Clotet. Systematic odometry error evaluation and correction in a human-sized three-wheeled omnidirectional mobile robot using flower-shaped calibration trajectories. *Applied Sciences*, 12(5), 2022.

[16] Murata Manufacturing Co., Ltd. Electromagnetic Interference from Industrial Equipment: Effects on Wireless Communication and Modern Mitigation Strategies. `https://article.murata.com/en-us/article/emi-industrial-equipment-impact-wireless-mitigation`, 2023.

[17] T. Zhang and V. Kumar. Robotic dispensing and sealing in industrial assembly processes. *IEEE Transactions on Automation Science and Engineering*, 2017.

[18] A. Bicchi and et al. Adaptive control for robotic manipulation of uncertain environments. *IEEE Control Systems Magazine*, 2000.

[19] S. Deshpande, A. Roy, J. Johnson, E. Fitz, M. Kumar, and S. Anand. Smart monitoring and automated real-time visual inspection of sealant applications (smart-vista). *Manufacturing Letters*, 35:1134–1145, 2023.

[20] L. D. Hanh and C. V. Thien. Visual guidance of a sealant dispensing robot for online detection of complex 3d-curve seams. *International Journal on Interactive Design and Manufacturing*, 16:1525–1532, 2022.

[21] F. R. Pereira, C. D. Rodrigues, H. da Silva e Souza, J. O. C. Neto, M. C. Rocha, G. F. Barbosa, S. B. Shiki,

and R. S. Inoue. Force and vision-based system for robotic sealing monitoring. *International Journal of Advanced Manufacturing Technology*, 126:391–403, 2023.

[22] J. Zhang, X. Yang, W. Wang, H. Wang, L. Ding, S. El-Badawy, and Z. You. Vision guided robot for automated pixel-level pavement crack sealing. *Automation in Construction*, 168:105783, 2024.

[23] J. Esteves and et al. Containerized robotic systems for industrial automation using ros and docker. *Robotics and Computer-Integrated Manufacturing*, 2021.

[24] M. Quigley and et al. Ros: An open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[25] E. Rohmer, S. P. N. Singh, and M. Freese. Coppeliasim: A versatile simulator for industrial robotic applications. In *IEEE/RSJ IROS*, 2013.

[26] Jonathan M. Aitken, Saeid Mokaram, Uriel Martinez-Hernandez, Iveta Eimontaite, David Cameron, James Law, Joe Rolph, and Ian Gwilt. KUKA-IIWA-API: Ros-enabled API for the KUKA LBR iiwa collaborative robot. `https://github.com/jonaitken/KUKA-IIWA-API`, 2017. GNU General Public License v3.0.