

Enhancing robotic interoperability: An approach for ROS 2-driven system integration

Seemal Asif^{a,*}, Mikel Bueno^a, Paul Anandan^b, Pedro Ferreira^b, Ze Zhang^c, Yue Yao^c, Windo Hutabarat^c, Gautham Ragunathan^d, Ashutosh Tiwari^c, Niels Lohse^e, Phil Webb^a, Lloyd L.C. Tinkler^d

^a Centre for Robotics and Assembly, Faculty of Engineering and Applied Sciences, Cranfield University, Cranfield, Bedfordshire, UK

^b Intelligent Automation Centre, Wolfson School of Mechanical, Electrical and Manufacturing Engineering, Loughborough University, Loughborough, UK

^c School of Mechanical, Aerospace and Civil Engineering, The University of Sheffield, Sheffield, UK

^d The University of Sheffield Advanced Manufacturing Research Centre (AMRC), Sheffield, UK

^e Birmingham Institute for Robotics, School of Engineering, College of Engineering and Physical Sciences, The University of Birmingham, Birmingham, UK

ARTICLE INFO

Keywords:

Robotic interoperability
ROS 2
Reconfigurable manufacturing systems (RMS)
Skill-based modelling
Model-driven robotics
AML integration
Intelligent perception systems

ABSTRACT

Reconfigurable manufacturing requires robotic workcells that can adapt to changing products, processes, and equipment without extensive redesign or manual reprogramming. This paper presents the R3M framework, a ROS 2 driven integration architecture that connects model based manufacturing knowledge with execution level robotic control. Product, process, and equipment information is formalised through UML domain models and serialised in AutomationML(AML), enabling automated correspondence between assembly requirements, available skills, and executable recipes. The framework integrates automated programme generation, reinforcement learning based recipe optimisation, and a modular CAD informed six degree of freedom perception layer to support both technical and semantic interoperability across simulated and physical workcells. The approach is evaluated through Cube Kitting and Cylinder Stacking use cases implemented on distinct robotic platforms, including ABB and Universal Robots systems. Experimental results show high reliability, with environment launch performance reaching up to 99.93%, standard skill sequences achieving 100% execution success, and full use case trials exceeding 97% success in simulation and reaching 100% on physical hardware. These findings demonstrate that R3M provides a scalable foundation for adaptive robotic manufacturing, reducing programming effort while supporting modular substitution, robust perception, and simulation to real deployment.

1. Introduction

The increasing demand for product customisation, short innovation cycles, and volatile market conditions is placing growing pressure on manufacturing systems to become more flexible, resilient, and responsive. These requirements align with the broader goals of Industry 5.0, which emphasise human-centred, adaptable, and sustainable production environments supported by intelligent automation. In this context, conventional robotic systems, although effective in stable and structured settings, remain limited by rigid architectures, task-specific programming, and difficult reconfiguration when products, layouts, or process requirements change [1].

Reconfigurable robotics has emerged as a response to these

limitations, promoting modularity, integrability, and scalability as key enablers of adaptive manufacturing. However, despite progress in reconfigurable manufacturing systems, robotic workcells still often require substantial manual effort to accommodate even modest changes. Updates in product variants, equipment availability, or cell layouts frequently lead to repeated modifications in task code, interfaces, and parameters. This creates a persistent gap between high-level manufacturing flexibility goals and the practical deployment of interoperable, reusable robotic cells.

R3M addresses this gap through a transferable approach for configuring and operating robotic workcells that combines a ROS 2-based execution backbone with structured domain knowledge describing products, processes, equipment, and skills. Its purpose is not only to

This research is supported by EPSRC grant (EP/V051180/1) for the Reconfigurable Robotics for Responsive Manufacture (R3M) Project.

* Corresponding author.

E-mail address: s.asif@cranfield.ac.uk (S. Asif).

<https://doi.org/10.1016/j.rcim.2026.103420>

Received 13 February 2026; Received in revised form 27 July 2026; Accepted 31 August 2026

Available online 15 September 2026

0736-5845/© 2026 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY license (<http://creativecommons.org/licenses/by/4.0/>).

connect heterogeneous robots and peripherals, but to support repeatable deployment and rapid reconfiguration while preserving alignment between simulation and real hardware execution.

At the execution level, R3M provides a reusable ROS 2-based workcell architecture integrating motion planning, control, simulation, and orchestration through standard interfaces. Formal modelling is used alongside this execution layer to represent products, processes, equipment, and skills in a consistent way, supporting a skill-based programming approach in which capabilities are described, matched, and reused through shared representations rather than repeatedly hard-coded for each new application.

Automated Program Generation (APG) is a central element of this approach. In R3M, executable robot programs are generated by composing reusable skills according to the current product specification, process requirements, and available equipment. This shifts engineering effort away from repeated low-level coding and toward structured models and validated skill libraries, reducing changeover effort and improving scalability across tasks and product variants.

Perception is treated in R3M as an enabling layer for object-centric task execution. Many robotic cells still rely on fixed spatial references and task-specific perception pipelines, which makes them brittle when parts, fixtures, or layouts vary [2]. In contrast, the R3M perception module provides object and state estimates that can be used directly by skills and APG, allowing tasks to be defined in terms of objects, relations, and desired states rather than absolute poses.

The main contribution of this work is therefore a semantically structured, ROS 2-driven interoperability framework for reconfigurable robotic manufacturing cells that unifies formal modelling, reusable execution interfaces, APG, and perception-supported object-state reasoning within a single architecture. In this way, the paper addresses not only the integration of heterogeneous robotic components, but the broader challenge of making robotic cells repeatably deployable, rapidly reconfigurable, and execution-ready across both simulated and real manufacturing environments.

The remainder of this paper is structured as follows. Section 2 reviews related work on reconfiguration, interoperability, and program generation in robotics and manufacturing. Section 3 introduces the proposed software architecture and its ROS 2-based implementation. Section 4 details the system integration approach for adaptive manufacturing. Section 5 demonstrates the framework through representative industrial use cases. Section 6 concludes with a discussion of findings, limitations, and directions for future research.

2. Related work

Research on reconfigurable robotics and adaptive manufacturing spans modular hardware design, scalable software architectures, perception systems, and automatic program generation (APG). While conventional robotic systems perform well in structured environments, they remain limited in adaptability, motivating reconfigurable manufacturing systems (RMS) that emphasise modularity, integrability, and scalability. Recent work has addressed these limitations through middleware-based interoperability, formal knowledge representation, and learning-enabled decision-making to support flexible and autonomous reconfiguration. Additional efforts focus on APG methods to reduce manual programming, real-time perception for environmental awareness, and modular hardware enabling physical reconfiguration of robotic cells. Together, these strands highlight progress toward flexible manufacturing systems while exposing gaps in semantic integration and unified system modelling, which this paper addresses through a UML-based, ROS 2-driven interoperability framework.

2.1. Reconfiguration in manufacturing and robotics software

2.1.1. Towards reconfigurability in modern manufacturing and industry

Modern manufacturing must adapt rapidly to fluctuating demands,

increased customisation, and technological evolution. RMS address this need by enabling quick changes in structure, hardware, and software to support evolving product and process requirements [3]. To be truly reconfigurable, systems must exhibit modularity, integrability, customisation, convertibility, scalability, and diagnosability, allowing efficient adaptation, monitoring, and expansion [4]. Reconfigurability in manufacturing is a broad and multi-faceted topic, with ongoing research addressing it from diverse perspectives ranging from early-stage design to real-time cell/system-level optimisation. Ongoing work in RMS spans various layers of abstraction, each tackling the challenge of adaptability in distinct ways.

At the system design level, Ontology-Based Engineering (OBE) approaches [5,6] enable reconfigurability by formalising domain knowledge and supporting automated decision-making during system conceptualisation. At the cell level, digital twin frameworks [7–9] enhance reconfigurability by allowing rapid evaluation of multiple layout and sensor configurations, often using genetic algorithms to optimise deployment under time and space constraints. Ergonomic optimisation [10] contributes to human-centred reconfiguration through adaptive fixtures that adjust to workers' physical needs, making collaborative workcells more flexible and efficient. In the context of circular manufacturing, modular robotic workcells [11] offer reconfigurable setups tailored for variable disassembly tasks, enabling responsiveness to diverse product geometries and uncertain processes.

Beyond the workcell, reconfiguration is being addressed at the shop-floor and factory level through smart factory architectures that integrate cyber-physical systems, semantic models, and digital twins to enable plug-and-produce adaptability [12]. These systems promote decentralised control and scalability, allowing production lines to self-organise in response to changing demands. Complementary approaches using cloud computing and AI enhance shop-floor responsiveness through predictive decision-making and coordinated control [9]. At the hardware level, self-reconfigurable robotic systems [13,14] offer a more granular approach to adaptability, with modular robots capable of autonomously altering their structure or behaviour to suit evolving tasks and environments, positioning them as key enablers of future-ready automation.

2.1.2. Software architecture for reconfigurable systems

Despite growing interest and advancements, current RMS implementations often remain context-dependent and lack a unified theoretical foundation. Future research is needed to explore formal architectural frameworks that can support scalable and modular RMS deployments across different domains. In particular, there is a notable gap in addressing the modularity of RMS at the architectural level, which could offer optimization strategies for module allocation and reuse across multiple product families [15]. How flexible and adaptive a production system can be, unquestionably depends on its software architecture and the level of modularity, flexibility and adaptability of its software components [16].

Several efforts have proposed generic and modular architectures to address the increasing complexity and interoperability needs in smart manufacturing. One approach derives a generic Industry 4.0 system architecture by combining strengths from various European projects, offering real-time communication, service orchestration, and data interoperability through middleware and data adapters [17]. This architecture aims to bridge fragmented, use case-specific systems by promoting flexibility and extensibility. Similarly, the HyProCell project implemented a modular architecture for hybrid production cells, demonstrating reconfigurability through service-oriented integration of CNC-based machines, legacy system retrofitting via OPC UA, and dynamic data exchange using MQTT and REST APIs [18]. More explicitly, a recent contribution presents a dedicated reconfigurable middleware software architecture tailored to support flexible production environments [19]. By formalising “reconfigurable quality attributes,” this work provides a quantifiable foundation for designing, evaluating, and deploying reconfigurable middleware platforms in real settings,

marking an important step toward standardised, testable architectures for future RMS.

2.1.3. ROS 2 as reconfigurable and modular architecture in robotics

ROS (Robot Operating System) is an open-source framework designed to support the development of flexible, modular robotic systems. It is particularly effective in scenarios requiring interoperability between heterogeneous hardware and software modules, helping overcome proprietary constraints tied to robot interfaces and communication protocols [20,21]. By enabling seamless integration of perception, planning, and control components, ROS facilitates rapid deployment and reconfiguration of robotic cells [22]. Its real-time communication capabilities and support for distributed architectures further reinforce its suitability as a middleware for RMS [23], making it an ideal foundation for modular, adaptive robot workcells.

ROS and ROS 2 have proven to be key in enabling modular and reconfigurable robotic systems across domains. *Robotframework* [24] utilises ROS to integrate diverse robotic skills through a plugin-based architecture, supporting easy adaptation across applications. *ChoiRobot* [25], built on ROS 2, facilitates distributed multi-robot coordination without a central controller, leveraging ROS 2's decentralised design. A modular ROS-based architecture for robotic workcells [22] demonstrates plug-and-produce integration and flexibility for rapid adaptation in manufacturing. These examples confirm ROS (and ROS 2, its second and improved version) as a robust middleware choice, simplifying integration and supporting scalable, reusable solutions aligned with reconfigurable system principles.

2.1.4. Comparative analysis of related architectures

To complement the previous discussion, Table 1 compares the proposed framework with representative architecture-level approaches from the literature. The comparison dimensions were chosen to reflect the main objectives of this work and were informed by comparative analyses adopted in recent studies on reconfigurable production

architectures and distributed robotic cell architectures [26,27].

The comparison shows that the reviewed works tend to emphasise different layers of the problem. Architectures addressing broader manufacturing-system reconfigurability, interoperability, and modular production integration usually provide stronger coverage of system-level concerns, but often address the robotic cell layer in less detail. Conversely, architectures centred on robotic workcells and ROS-based integration tend to provide stronger support for robot abstraction, perception, and simulation-oriented deployment, but are often validated in narrower or more application-specific settings. Overall, the comparison suggests that the main contribution of R3M lies in bringing these two perspectives together within a single framework, unifying system-level reconfigurability concerns with robotic-cell-level integration capabilities in a coherent ROS 2-driven architecture.

Among the architectures included in Table 1, [19,22,25] provide three particularly relevant and complementary architectural points of comparison with R3M. First, [19] focuses on factory-level Industry 4.0 middleware and on specifying and evaluating reconfigurability as a software quality attribute. Second, [22] addresses modular ROS-based robotic workcells, with particular emphasis on hardware and software plug-and-produce integration, kinesthetic teaching, and high-level task programming. Third, [25] focuses on decentralised ROS 2 communication and distributed optimisation for cooperative multi-robot systems. R3M differs from these approaches by integrating formal product, process, equipment, and skill representations with automatic programme generation, reinforcement-learning-based recipe selection, modular perception, and ROS 2-based execution, thereby providing a semantic pathway from manufacturing requirements to executable and verifiable robotic behaviour.

2.2. Automatic program generation (APG)

APG has evolved from rule-based and manually coded systems toward more adaptive approaches for generating robotic control logic,

Table 1

● Full Support, ● Partially Covered, ● Not Covered, (S) Single Use Case, (M) Multiple Use Cases. Comparison of the proposed R3M architecture with representative architecture-level approaches discussed in the related work. The comparison considers key dimensions relevant to reconfigurable robotic and production-system architectures, including software and hardware reconfigurability, interoperability, scalability, modularity, plug-and-play capability, semantic/model-based representation, perception integration, ROS/ROS 2 implementation, robot layer abstractions, simulation-to-real transition capability and support, human-in-the-loop support, and validation scope. (References ([7],[16],[17],[18],[19],[22],[24],[25]) is cited in Table body part).

Reference	Reconfigurable Software Architecture	Reconfigurable Hardware Cell Support	Interoperable Architecture	Scalable Architecture	Modular Architecture	Plug-and-Play Capability Plug-and-Produce Capability	Semantic/Model-based Representation	Perception Integration/Capability	ROS/ROS 2-based Implementation	Robot Layer Abstractions	Simulation-to-Real Transition Capability and Support	Human-in-the-Loop	Validation Scope
[7]	●	●	●	●	●	●	●	●	●	●	●	●	S
[16]	●	●	●	●	●	●	●	●	●	●	●	●	S
[17]	●	●	●	●	●	●	●	●	●	●	●	●	M
[18]	●	●	●	●	●	●	●	●	●	●	●	●	S
[19]	●	●	●	●	●	●	●	●	●	●	●	●	S
[22]	●	●	●	●	●	●	●	●	●	●	●	●	S
[24]	●	●	●	●	●	●	●	●	●	●	●	●	M
[25]	●	●	●	●	●	●	●	●	●	●	●	●	M
R3M	●	●	●	●	●	●	●	●	●	●	●	●	M

reflecting the need for efficient and flexible programming methods in dynamic manufacturing environments.

Early work focused on logic-based planning and static task decomposition. Medellin et al. [28] used octree-based reasoning to co-plan human and robot assembly operations, while Ritchie et al. [29] presented a generative assembly planner based on geometric constraints. Although effective at the planning level, these approaches offered limited interoperability with equipment-level capabilities and control interfaces. Later methods sought to bridge this gap by deriving motion plans from CAD models or demonstrations. PathML, based on AutomationML (AML), generated CAD-derived paths for applications such as glass adhesive deposition and welding [30], while CAD-based systems have also been used to automatically generate welding programs with high adherence to good practice [31].

At the same time, the skill-based paradigm reintroduced hierarchical abstraction into robot programming. AML has played an important role as a cross-domain representation standard [32], while Lohi et al. [33] and the openMOS ecosystem [34] showed how parameterised and semantically described skills can support reuse across manufacturing settings. These studies demonstrated the value of modularisation and shared representations, but also revealed the need for richer hierarchical knowledge structures and more systematic skill composition for complex tasks.

Another persistent challenge in APG is parameter selection. Inoue et al. [35] demonstrated learned policies for precision tasks such as peg-in-hole insertion without exhaustive parametric enumeration. More broadly, recent APG approaches have used reinforcement learning for decision-making and parameter tuning [35,36], but few frameworks combine recipe generation, learning-based selection, and executable code deployment within a single workflow.

More recently, LLM-based and generative AI approaches have been explored for robotic code generation. MacAluso et al. [37] proposed ChatGPT-driven task decomposition into robot code with simulated debugging, while RoboGPT and related systems investigated LLM-guided sequence generation from abstract task descriptions [38]. Although promising, these approaches are often domain-specific, rely on engineered prompts or templates, and remain weakly connected to simulation-backed validation and real-world industrial execution.

Overall, the literature shows substantial progress in APG, but persistent limitations remain in scalability, adaptability, and deployment readiness. R3M addresses these gaps by combining domain modelling through AML, scalable skill parameterisation through composite recipes, reinforcement-learning-based recipe selection, and ROS-based generation of deployable execution logic within a unified framework.

2.3. Perception systems

Due to the high adaptivity and dynamic flexibility, vision-based perception plays an important part for reconfigurable robot cells. It enables robots to recognise and adapt to variable conditions in real-time, such as changes in workpieces, which can be essential for manufacturing applications. In the literature, there are a number of research works that attempt to estimate or track the spatial position and orientation of the manipulated object by using visual sensors. Among these studies, the deep learning approach can be a dominant method due to its high accuracy, robustness, and ability to extract high-level features in raw visual data which can be dense and complex -> Vision-based perception is critical for reconfigurable robotic cells, enabling real-time adaptation to variable manufacturing conditions, while deep learning has emerged as a predominant approach for robust and accurate object pose estimation from complex visual data [39–41]. Initial attempt in pose estimation have made towards the conventional learning-based methodologies, wherein the estimation models are tailored to predict the pose of predefined objects. This process necessitates the assembly of a comprehensive dataset of the objects of interest, serving as the training dataset.

Upon successful training, the model can estimate the pose of objects represented within the dataset. The combination of Perspective-n-Point (PnP) and deep learning can be a popular category of methods in this type of training paradigm. Given a set of n 3D points on the object and their corresponding 2D projections in an image, the 3D pose that best matches the 2D-3D point correspondences, typically under perspective projection, can be obtained. A representative PnP method can be found in [42,43]. For texture less objects, establishing accurate 2D-3D correspondences becomes more challenging due to the lack of distinct features to match. To address this problem, in [44], the authors leveraged the learning capacity of deep neural networks for direct pose estimation from 2D images and achieved a benchmark in this domain.

However, the above-mentioned methods required that the 3D models of objects have to be available during both training and inference, which limits their applications in many manufacturing settings, where the manipulated objects can change frequently due to the rapid change of market needs and production requirements. To address this challenge brought by real world applications and generalise the pose estimation to novel objects, some work in the literature attempts to establish pipelines for training models using synthetic data, thus allowing pose estimation to be applied to any novel objects, such as [45]. There are some other research works that try to establish the category-level estimation to handle the novel objects within a known category, such as [46–48]. However, the former often consumes a significant amount of time to generate synthetic data and retrain the model, and the latter strictly requires that the novel object can be included by a known class. Since, firstly, down time in industrial production causes efficiency loss as well as resource wastage, secondly, many middleware in industrial production can be very difficult to be classified to pre-defined categories, both approaches might be suboptimal. Recently, a new training paradigm is attracting the attention of robot perception and computer vision researchers. This paradigm attempts to enable deep learning models to acquire the ability to find actual objects in the vision field that correspond to the given 3D models by performing feature extraction on both the 3D model of the object and the visual signals during the training process. By training with a huge number of 3D models with paired vision observations, given any 3D model, the deep learning model is able to estimate its pose. This allows the generalisation ability of the estimation model to be greatly improved. In the manufacturing industry, where CAD models of workpieces can often be determined during the product design phase, this training paradigm has great potential to adapt to the frequently changing product designs and market demands in the manufacturing industry. A representative research breakthrough using this paradigm can be found in [49].

2.4. Reconfigurability in hardware for industry and manufacturing

There are a number of approaches that can be applied to hardware design to support the realisation of a reconfigurable robotic manufacturing cell. The key challenge is to ensure the system is modular and composed of a series of 'building blocks' and can be rearranged into different configurations [50]. Interfaces between the modules should be standardised so that different components or components from different manufacturers can be interchanged to perform a specific process. One of the most important modules in a robotic cell is the end effector and a huge range of use cases can be fulfilled just by installing a tool changer on the end of a robot arm and having a range of end effector options such as grippers (electric, magnetic, pneumatic), process specific tools (such as gluing, prying, screwing, cutting and grinding), sensors and vision systems.

Outside of the robot itself, the surrounding hardware can also be designed to demonstrate a reconfigurable robotic cell. A reconfigurable floor with an array of mounting plates with different mounting hole patterns enable various hardware (robots, fixtures, parts etc.) to be moved physically around the cell to demonstrate use cases with different space and access requirements. One such example of a reconfigurable

floor is the AMRC's RECON Cell in the Factory 2050 facility which was used as the demonstrator cell for this project (see Section 4.4. Hardware Deployment). Additionally, fixtures form an essential part of holding component parts in place for a variety of industrial applications.

Designing flexible and reconfigurable fixtures can greatly reduce the costs associated with bespoke fixtures for single components, especially in the Aerospace industry where tolerances are tight. There are different types of fixtures that can handle a family of components. Zero-point clamps can be used to clamp a workpiece directly or to clamp a pallet containing many workpieces. They can have different levels of automation and can be locked in using different methods (pneumatics, electric, hydraulic etc.). Another type of flexible fixture is a universal holding fixture (UHF). These are an array of height adjustable vacuum grippers that can be reconfigured to conform to the surface geometry of a part. These elements are typically mounted on extendable linear rails or linear actuators to provide vertical adjustment and are typically used for low force machining and drilling tasks on thin and flexible components.

In a real industrial environment, all of the hardware in the cell must be controlled by robust industrial control systems such as PLCs, IPCs and safety hardware. The control architecture must be capable of potentially complex and interlinked use cases with different sensors, robots, and fixtures required for each use case and be able to detect the presence (or absence) of any key hardware components. There must be wiring access to appropriate input and output blocks around the cell as well as peripheral hardware requiring pneumatics or hydraulics. Finally, the safety architecture must be tested to ensure safe operation of the various use cases and configurations of the cell.

2.5. Discussion

The reviewed literature demonstrates considerable progress in advancing reconfigurability and interoperability in robotics, spanning RMS, middleware frameworks, APG, perception systems, and modular hardware design. Collectively, these strands underscore the importance of modularity, scalability, and adaptability, yet they also reveal persistent shortcomings that hinder the realization of autonomous, reconfigurable robotic cells. A recurring limitation is the context dependency of existing solutions. While RMS research has established modularity and integrability at various levels [2], these approaches often lack a unifying architectural foundation to ensure semantic consistency across domains. Similarly, middleware such as ROS 2 [19] enables communication among heterogeneous systems but generally requires extensive customisation and does not inherently embed domain knowledge, thereby achieving technical rather than semantic interoperability. APG has shown the potential to reduce programming effort [36,51], though many implementations remain bound by domain-specific assumptions and weak integration with diverse equipment. Advances in perception [37–43] enhance adaptability to dynamic environments, yet most approaches remain constrained by task-specific datasets and limited generalization, while hardware-level reconfigurability [48] provides flexibility but remains insufficiently integrated with higher-level semantic models.

Despite these advances, the literature remains fragmented, with different research streams addressing isolated aspects of the reconfigurability challenge without converging into a holistic solution. Middleware-centric approaches prioritise communication, perception-focused solutions emphasise sensing, and hardware research advances modularity, but none provide an integrated pathway from high-level domain abstractions to executable robotic behaviour. To address this gap, the present work proposes a UML-based, ROS 2-driven integration framework that unifies domain modelling with execution-level mechanisms. By embedding APG, modular perception, and reinforcement learning within a shared semantic structure, the framework offers not only technical but also semantic interoperability. The novelty of this approach lies in bridging formal UML-based representations with real-

time robotic execution through ROS 2, thereby establishing a principled foundation for scalable, interoperable, and reconfigurable robotic manufacturing. The following section details the architecture underpinning this contribution.

3. Software architecture

This section presents the software architecture that operationalises the interoperability objectives of R3M through a reusable ROS 2-based workcell architecture, referred to as the R3M Cell, and a standardised operational interface for external integration. The architecture is designed so that simulated and real robot cells are handled through the same execution structure, with robot- and device-specific details isolated behind shared representations and standard ROS 2 communication interfaces. This reduces commissioning effort, improves portability across heterogeneous platforms, and provides the execution basis required by higher-level functions such as APG and modular perception.

3.1. ROS 2-based architecture: the R3M cell

The R3M Cell (Fig. 1) provides a reusable ROS 2-based architecture for the simulation and operation of robotic workcells. Its central principle is that, from the perspective of planning and execution, both simulated and real robot cells are handled through the same abstraction layer. This is achieved through a shared robot representation, based on standard ROS 2 description formats such as URDF, which captures the kinematic, geometric, and control-relevant structure of the robot in a common form.

This shared representation is not only what enables alignment between simulation and hardware execution, but also what allows robots from different manufacturers to be integrated and operated through the same ROS 2 and MoveIt!2 control layer. Once a robot is described in this common format and connected through the appropriate driver or adapter, it can be planned, monitored, and commanded using the same operational logic regardless of vendor-specific implementation details. In this sense, the robot is treated uniformly by the upper execution layer whether it is simulated in Gazebo or connected as physical hardware.

Within this structure, MoveIt!2 provides the common motion-planning and execution interface [52], while Gazebo supports simulation and the ROS 2 driver connects the same architecture to the real robot cell. The distinction between simulated and real deployment is therefore pushed to the lower execution layer, while the upper operational logic remains unchanged. At the same time, the use of a standard robot representation makes it possible for different manipulators and peripheral devices to be incorporated into the same workcell architecture without redefining the overall execution structure.

The main contribution of the R3M Cell is therefore not the introduction of new simulation or planning tools, but the definition of a unified and reusable workcell architecture that combines them within a consistent ROS 2 execution framework. By encapsulating robot- and device-specific details in adapters and shared robot specifications, the architecture supports redeployment across different manipulators and peripheral devices while maintaining alignment between simulation and hardware execution.

3.2. Interoperability framework for modular robotics

Built on top of the R3M Cell, the interoperability framework provides a standardised operational layer for robot control, monitoring, and external integration. Instead of exposing users or applications to the full complexity of MoveIt!2 and low-level ROS 2 execution, the framework encapsulates key functionalities through stable ROS 2 communication mechanisms called topics, services, and actions.

Robot state monitoring is exposed through dedicated topics for joint states and end-effector pose. Motion execution is handled through standardised action interfaces that support different movement types

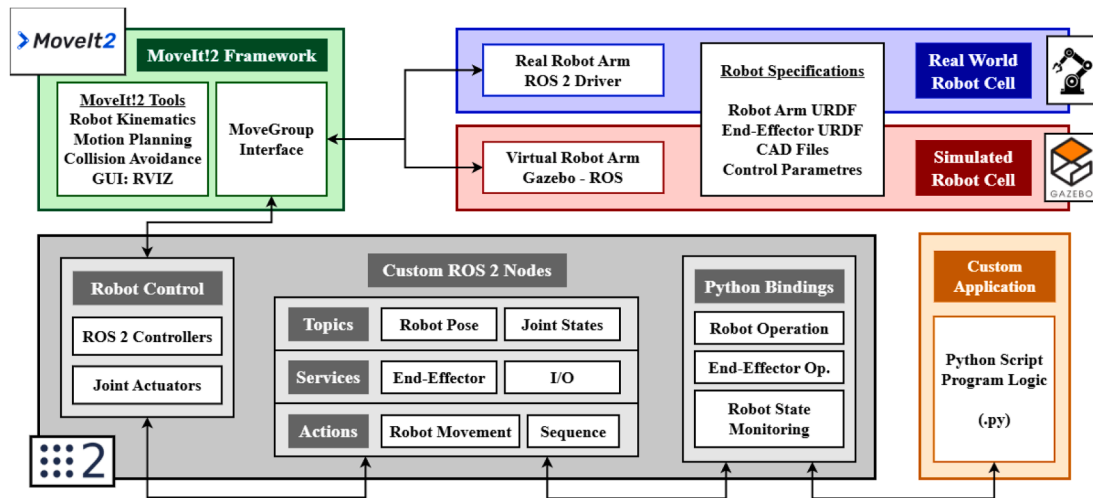


Fig. 1. Unified ROS 2-based architecture of the R3M Cell. A common robot representation is shared across simulation and real robot deployment, allowing both to be handled through the same planning and execution structure. The lower part shows the interoperability framework, where standard ROS 2 topics, actions, and services, together with Python bindings, provide a robot-agnostic interface for control, monitoring, and external integration.

and sequencing operations. Services are used for peripheral interaction, such as end-effector actuation and I/O operations. This creates a robot-agnostic operational interface that can be reused across different robotic cells without dependence on vendor-specific APIs [53,54].

To further simplify integration, a lightweight layer of Python bindings is provided for external applications. These bindings allow higher-level software to command robot movement, retrieve state information, and operate peripherals through concise and consistent function calls, avoiding direct manipulation of ROS 2 message structures. As a result, the interoperability framework supports modular integration with planning, optimisation, supervision, and application-specific logic, while preserving a stable execution interface across heterogeneous robots and peripherals.

Together, the R3M Cell and the interoperability framework establish a common operational foundation for modular robotic workcells, enabling simulation and hardware deployment to be handled consistently and allowing robots from different manufacturers to be integrated and operated through the same ROS 2-based execution model.

4. System integration for adaptive manufacturing

This section examines the integration of the ROS 2 based stack and the R3M Cell, detailing how modular capabilities are composed into a dependable production system. Each capability is formalised as a skill with explicit interface contracts, preconditions, effects, and quality-of-service guarantees. A shared semantic model aligning product, process, and equipment enables requirements to be transformed into machine-readable recipes. These specifications drive an APG pipeline that selects and composes skills into executable task graphs, while a lightweight runtime orchestrator coordinates perception, program execution, certified PLC safety functions, field I/O, and Human Machine Interface (HMI). Perception modules integrate through stable message schemas and reference frames, and deterministic handshakes across action servers and field buses preserve temporal predictability. Continuous feedback from equipment and vision maintains a coherent world state, supporting fault recovery and adaptation without architectural refactoring. The resulting integration yields shorter commissioning, higher reuse across heterogeneous hardware, clear separation of concerns, and end-to-end observability traceable to design intent.

4.1. Requirements definition for responsive robotic cell configuration

A key component of the APG methodology in the R3M project is the

structured representation of information required for automated robot programming. The effectiveness of the approach depends on how well the system captures and connects knowledge across three domains: Product, Process, and Equipment. These domains are formalised using UML class models to ensure a consistent and machine-interpretable structure.

4.1.1. Product domain

The Product domain describes the assembly requirements of the product. A product is composed of parts or subassemblies, each associated with skill requirements as shown in Fig. 2. These skill requirements define the operations needed to complete the assembly. A key feature of this model is the precedence relationship between skill requirements. This defines the order in which assembly steps must be executed. The product domain also introduces the concept of a liaison, which represents the physical connection between parts. Although geometric details are not explicitly defined, liaisons capture how parts interact and provide a basis for evaluating assembly completion. Each skill requirement is associated with a skill type as shown in Fig. 2. This shared attribute is essential, as it enables automated matching with process and equipment capabilities.

4.1.2. Process domain

The Process domain defines how assembly tasks can be executed. It is centred around the concept of composite recipes as shown in Fig. 2. The composite recipes represent reusable task structures derived from human operational expertise. Each composite recipe consists of one or more skill requirements; each linked to a skill type. This enables the system to map product requirements to possible execution strategies. The requirements definition model supports multiple alternative recipes for a given requirement. These alternatives may differ in performance characteristics such as execution time, energy usage or precision. This flexibility is essential for later optimisation stages.

4.1.3. Equipment domain

The Equipment domain captures the capabilities of the robotic cell. It includes physical components such as robots, grippers, and sensors, along with their associated skills as shown in Fig. 2. Each skill is defined with input and output parameters. These parameters describe how the skill can be executed under different conditions. Possible parameter settings define the range of configurations for each skill. In addition, performance-related information such as execution time can be associated with each skill. This information is used later during optimisation.

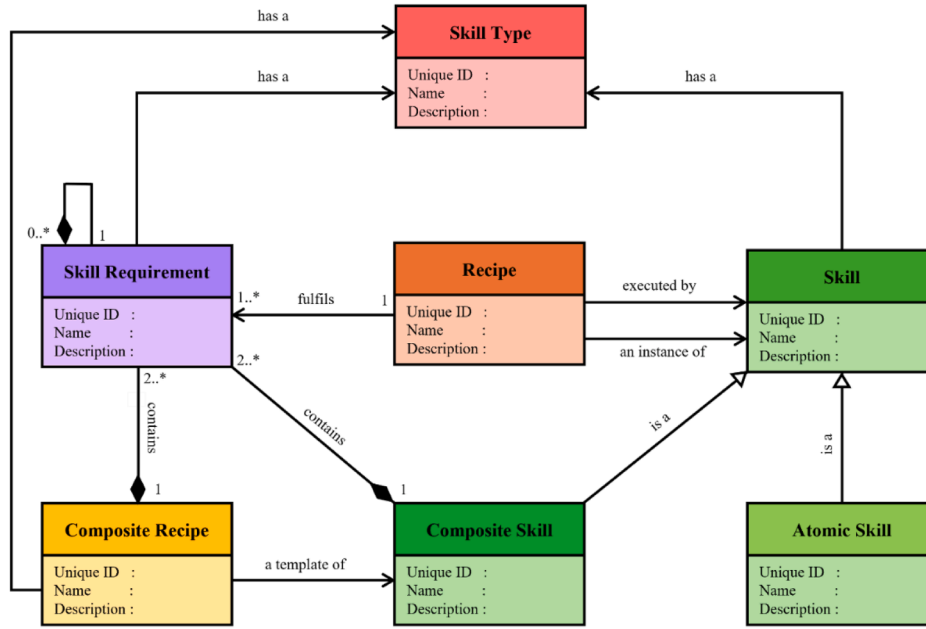


Fig. 2. UML based domain model for requirements definition in automated robotic programme generation. The model formalises the relationships between product skill requirements, process recipes, and equipment skills through a shared Skill Type abstraction, enabling systematic matching between assembly intent, reusable task structures, and available execution capabilities within the robotic cell.

By linking each skill to a skill type, the model enables consistent matching between product requirements, process recipes, and equipment capabilities.

4.2. AI-driven program generation for flexible automation

The APG methodology developed in the R3M project offers a structured approach for autonomously creating executable programs for robotic assembly cells. The goal is to minimise human intervention by using existing product, process, and equipment models, combined with intelligent reasoning and optimization techniques. The methodology unfolds through three core phases: recipe generation, recipe optimization, and code generation.

4.2.1. Recipe generation

The recipe generation phase translates product requirements into executable robotic actions. A recipe is defined as a parameterised instance of a skill that fulfils a specific assembly requirement. All required information is obtained from the AML model, which integrates the product, process, and equipment domains.

The key mechanism enabling this phase is the shared attribute Skill Type. This attribute is common to:

- Product skill requirements.
- Composite recipes.
- Equipment skills.

It provides a direct basis for automated matching across domains.

For each product skill requirement, the system first identifies candidate composite recipes whose skill types match the requirement. Composite recipes represent predefined task structures derived from expert knowledge. A single requirement may be satisfied by multiple composite recipes, each representing a different execution strategy.

Each selected composite recipe is then decomposed into its internal skill requirements. These are matched against available equipment skills, again using the skill type attribute. A valid mapping is established only if all internal skill requirements can be fulfilled by the equipment. Once a valid composite recipe–equipment mapping is identified, the

system proceeds to parameterisation. Skill parameters define how the action is executed. For simple skills, parameters are limited (e.g., open/close). For motion skills, parameters include pose, orientation, and speed.

This parameterisation step introduces a combinatorial challenge. A single action may have multiple valid configurations. To control this, the system generates only feasible parameter combinations, based on:

- AML-defined constraints.
- Task-specific conditions.
- Safety checks (e.g., collision, range limits).

Invalid or unsafe combinations are discarded. Valid configurations are instantiated as recipes and added to the final recipe set. The process is applied iteratively for each product skill requirement, resulting in a structured set of executable recipes.

The output of this phase is a set of parametrised recipes. Each recipe represents a feasible way to fulfil a product requirement using available equipment. These recipes form the input for the subsequent reinforcement learning phase. The source code of the skill requirements to recipe generation process is shown in [Pseudo-code 1](#).

4.2.2. Reinforcement learning

Following the generation of parameterised recipes, the next phase aims to determine an optimal sequence of recipes that fulfils the product assembly requirements while minimising execution cost. This optimisation problem is formulated as a Markov Decision Process (MDP), defined by the tuple $M = (S, A, P, R, \gamma)$, where S is the state space, A is the action space, $P(s_{t+1}|s_t, a_t)$ is the transition probability, $R(s_t, a_t)$ is the reward function, and γ is the discount factor.

At each time step t , the system state $s_t \in S$ is represented as a combined product, equipment, and process state:

$$s_t = [s_t^{Prod}, s_t^{Equip}, s_t^{Proc}] \quad (1)$$

where s_t^{Prod} contains the poses of the relevant parts and the status of the liaisons required for assembly completion, s_t^{Equip} contains the robot pose and the end-effector state, and s_t^{Proc} contains the current progress of the

Pseudo-code 1

Recipe Generation.

```

for each skill_requirement in product:
    candidate_recipes = find_composite_recipes
    where recipe.skill_type == skill_requirement.skill_type
    for each composite_recipe in candidate_recipes:
        sub_skills = composite_recipe.skill_requirements
        matched_skills = []
        for each sub_skill in sub_skills:
            equipment_skill = find_equipment_skill
            where equipment_skill.skill_type == sub_skill.skill_type
            if equipment_skill exists:
                matched_skills.add(equipment_skill)
            else:
                discard composite_recipe
        continue outer_loop
    parameter_sets = generate_parameters(matched_skills, AML_constraints)
    for each params in parameter_sets:
        if is_feasible(params):
            recipe_instance = create_recipe(composite_recipe, params)
            add recipe_instance to recipe_set
        else:
            discard params

```

assembly process and the remaining skill requirements to be satisfied. The state is updated after each executed action using feedback from the digital twin or physical robotic cell.

The action $a_t \in A$ corresponds to the selection of a recipe ID from the recipe set generated during the recipe generation phase. Each recipe ID represents a parameterised skill instance, such as a movement, grasping, release, inspection, or placement operation. The action space is therefore discrete and finite:

$$A = \{r_1, r_2, \dots, r_n\} \quad (2)$$

where r_n denotes the n^{th} candidate recipe available to the agent. The transition function is realised through the digital twin environment, which simulates the execution of the selected recipe and returns the updated system state. The reward function is formulated as a weighted penalty function that captures the cost of executing a selected recipe. In its general form, the reward at time step t is defined as:

$$R(s_t, a_t) = -(\omega_T T_t + \omega_E E_t + \omega_R R_t + \omega_H H_t + \omega_F F_t) \quad (3)$$

where T_t represents the execution time of the selected recipe, E_t represents the energy consumption, R_t represents a penalty for excessive product rotation during handling, H_t represents a handling stability penalty, and F_t represents a failure-related penalty. The weighting factors ω_T , ω_E , ω_R , ω_H , and ω_F allow the formulation to prioritise different performance objectives depending on the production context.

For the experimental evaluation reported in this work, execution time is used as the dominant optimisation criterion. The reward therefore reduces to:

$$R(s_t, a_t) = -T_t \quad (4)$$

This means that recipes requiring longer execution times receive larger penalties, while faster recipes produce higher rewards. The objective of the agent is to learn a policy $\pi_\theta(a|s)$ that maximises the expected discounted return:

$$J(\theta) = \mathbb{E}_{\pi_\theta} \left[\sum_{t=0}^T \gamma^t R(s_t, a_t) \right] \quad (5)$$

Since the reward is penalty-based, maximising $J(\theta)$ is equivalent to minimising the cumulative execution time of the selected recipe sequence.

The policy is learned using Proximal Policy Optimization (PPO), a policy-gradient method selected for its stable policy updates and suitability for discrete action-selection problems. PPO optimises a clipped surrogate objective, which limits large policy updates between training

Table 2

PPO actor-critic network architecture used for APG recipe selection.

Component	Layer	Size / Output	Activation	Purpose
Actor	Input layer	n_s neurons	None	Receives the APG system state vector.
Actor	Fully connected layer 1	128 hidden units	ReLU	Extracts state features.
Actor	Fully connected layer 2	128 hidden units	ReLU	Forms a higher-level state representation.
Actor	Output layer	n_a neurons	Softmax	Produces a probability distribution over recipe IDs.
Critic	Input layer	n_s neurons	None	Receives the APG system state vector.
Critic	Fully connected layer 1	128 hidden units	ReLU	Extracts state features.
Critic	Fully connected layer 2	128 hidden units	ReLU	Forms a higher-level state representation.
Critic	Output layer	1 neuron	Linear	Estimates the state value $V(s_t)$

iterations and improves stability during learning. This is appropriate for the APG problem because the agent must repeatedly select one action from a finite set of generated recipe IDs. PPO was introduced by Schulman et al. as a practical policy-gradient method that alternates between sampling data through environment interaction and optimising a surrogate objective over multiple epochs.

To improve reproducibility, the PPO agent is implemented using an actor-critic architecture. The actor network receives the system state vector as input and outputs a probability distribution over the available recipe IDs. The critic network receives the same state vector and estimates the corresponding state value $V(s_t)$. Both networks use fully connected layers with nonlinear activation functions. The actor output layer uses a softmax activation to represent the discrete probability distribution over candidate recipe actions, while the critic output layer uses a linear activation to estimate the scalar value function.

The actor and critic network architectures are summarised in Table 2. The PPO training hyperparameters are summarised in Table 3. These tables should be populated using the exact values from the MATLAB/Simulink PPO implementation used in the final experiments.

Here, n_s denotes the state-vector dimension and n_a denotes the number of discrete recipe actions available to the agent.

The PPO agent was trained using the Adam optimiser with a learning rate of 1×10^{-4} . A discount factor of 0.99 was used to encourage the agent to optimise the full assembly sequence rather than isolated recipe selections. The PPO clip factor was set to 0.20 to restrict large policy updates and improve training stability. The entropy loss weight encouraged early exploration across alternative recipe choices, while the value function loss weight-controlled critic learning. Training was terminated when the agent consistently achieved the minimum cumulative execution-time penalty over five consecutive successful episodes. The trained policy is then used in the subsequent code generation phase to produce executable robot programs.

4.2.3. Code generation

The final phase converts the learned policy into an executable robot program, enabling deployment on real hardware. The trained PPO agent is exported using MATLAB's code generation tools and packaged as a Python module via MATLAB Compiler SDK. Execution in a Linux-based environment is supported through the MATLAB Runtime, allowing royalty-free deployment without a full MATLAB installation.

At runtime:

Table 3

PPO training hyperparameters used in the APG reinforcement learning implementation.

No.	Hyperparameter	Value
1	RL algorithm	Proximal Policy Optimization (PPO)
2	Observation/state dimension	n_s , use-case dependent
3	Number of discrete actions	n_a , equal to number of generated recipe IDs
4	Actor hidden layers	2 fully connected layers
5	Critic hidden layers	2 fully connected layers
6	Hidden units per layer	128
7	Hidden activation function	ReLU
8	Actor output activation	Softmax
9	Critic output activation	Linear
10	Optimiser	Adam
11	Learning rate	1×10^{-4}
12	Discount factor gamma	0.99
13	Generalised advantage estimation factor	0.95
14	PPO clip factor	0.20
15	Entropy loss weight	0.01
16	Value function loss weight	0.5
17	Mini-batch size	64
18	Experience horizon	512 steps
19	Number of epochs per update	3
20	Maximum training episodes	Use-case dependent
21	Episode termination condition	Successful task completion or failure condition
22	Training termination condition	Five consecutive successful episodes with minimum cumulative execution-time penalty

- The system provides a state input.
- The trained agent outputs an action (recipe ID).
- The selected recipe is executed by the robotic system.

This establishes a direct mapping from system state to recipe selection. Each recipe is translated into manufacturer-specific robot code using predefined templates (e.g., RAPID, KRL, URScript). Parameters such as pose, speed, and gripper state are injected into these templates, preserving execution order and constraints. The result is an executable system combining:

- a Python-based decision module, and
- a robot-specific execution layer.

This enables seamless deployment and rapid reconfiguration without manual programming.

4.3. Sensor-driven perception systems for intelligent decision-making

As part of a modular and interoperable robotic framework, the perception system is designed to operate as a sensor-driven, decoupled module within the broader R3M architecture, enabling seamless interoperability and flexible system configuration. The system's modular design ensures that the perception component can be easily swapped or reconfigured with alternative algorithms, allowing developers to tailor perceptual capabilities to specific application requirements without modifying the core architecture.

4.3.1. Perception system features

In its current form, this module processes visual input to estimate 6DoF poses of previously unseen objects in dynamic manufacturing environments, aiming at introducing the adaptivity in order to adapt to the dynamic and versatile nature of modern manufacturing environments. This framework integrates advanced vision-based *One-Shot Detection* (OSD) [55] and *MegaPose-6D* (M6D) pose estimator [49] to enable the precise identification and spatial localisation of novel objects

using their CAD models as the reference. Contrary to traditional deep learning approaches, which are typically constrained by the scope of their training datasets, limiting their utility to the detection and tracking of previously encountered objects, the deep learning models employed in this perceptual system were developed through an extensive training involving a diverse array of CAD models paired with visual observation. This methodological innovation enables the models to match the geometric properties embedded within the CAD files to their real-world counterparts, effectively allowing for the accurate detection and estimation of the position and pose of objects in the physical environment, even if these specific instances were not included in the training dataset. Consequently, such a training paradigm shift significantly enhances the system's applicability, as it no longer requires that the workpieces or its categories have been seen during the training phase, accommodating rapid changes in product designs and market need without the need for retraining the models, thus representing a substantial advancement in the field of robotic perception. -> The pose estimator models are trained entirely on synthetic RGB-D data generated with BlenderProc, using approximately two million photorealistic images from ShapeNet and Google Scanned Objects with ground-truth 6D pose annotations and corresponding 3D models. To improve generalisation to novel objects and support sim-to-real transfer, object placement, materials, backgrounds, lighting, and camera viewpoints are randomised, while RGB and depth augmentations are applied following established approaches such as CosyPose. The models are trained once using 32 V100 GPUs, requiring approximately 80 h, after which novel-object pose estimation can be performed without target-specific fine-tuning.

4.3.2. Integration within R3M

The system architecture comprises two main components: the One-Shot Detection module [55] and a novel-object pose-estimation module [49]. The One-Shot Detection module identifies objects within the camera's field of view using visual data and CAD models as inputs and generates corresponding bounding boxes. The bounding-box coordinates, camera observation, and CAD model are then passed to the pose-estimation module, which estimates the 6DoF pose of each object. Accurate pose estimation requires the target object to occupy a dominant portion of the viewing field; therefore, the bounding box does not need to have precisely estimated dimensions, but it must localise the object sufficiently tightly for the object to occupy most of the cropped region. To improve reliability, the system incorporates a human-in-the-loop validation mechanism that allows operators to verify or correct the bounding boxes through an intuitive HMI. Further details of the perception workflow and the interactions among the R3M architecture, HMI, operator, camera, detection module, and pose-estimation components are presented in [Appendix A](#).

The process begins with the R3M architecture triggering the One-Shot Detection component, followed by requesting a single frame from the camera by the shot detection. The resulting bounding box will then be sent to the HMI for an operator to verify. If the bounding box is valid, it is sent to the MegaPose-Coarse Estimator for initial 6DoF pose estimation. Otherwise, the operator can manually provide a bounding box via HMI by clicking and dragging for the downstream actions, as shown in [Fig. 3](#).

Next, a video stream is requested to enable the MegaPose-Refiner to refine the pose estimation, which is an iterative process. Each iteration of the refinement process takes the previous pose estimate as input and produces a more accurate output. This iterative loop continues until a predefined termination condition is met, either when convergence is reached or after a set number of iterations. The resulting final pose, which reflects a high degree of spatial precision, is then published for downstream robotic manipulation, augmented reality, or scene understanding applications. This combination of human oversight, fast detection, and fine-grained refinement enables the pipeline to robustly handle objects not seen during training, making it well-suited for dynamic, real-world environments.

4.4. Hardware deployment

The R3M architecture is the software framework that enables simulation and control of robotic systems, but in order to demonstrate real world use cases, it must be deployed on actual hardware. The architecture has been tested and deployed on multiple cells and laboratories with various use cases and reconfigurations. In this section, the general structure of the hardware deployment is explained and the best practices and requirements for safe operation in an industrial environment are presented.

The hardware overview for deploying the R3M Architecture in an industrial cell is shown in Fig. 4. The PLC acts as the master device for the entire cell and is responsible for cell safety and monitoring of the safety states of the hardware. It should use a fieldbus protocol (such as EtherCAT, PROFINET etc.) for communication with industrial devices and sensors allowing for real time control, standardization and ensuring safety. The R3M software architecture is hosted in a separate environment to the PLC such as on its own dedicated industrial PC (as shown in Fig. 4) or if the cell is set up with an Industrial PC running a soft PLC, then the R3M architecture can be installed on a Ubuntu Virtual Machine (VM) hosted on the same device. This separation between the PLC and the R3M architecture is deliberate and necessary for safety-critical environments. The R3M architecture can only control the robot motion and when the cell is set up for remote control of the robot. In practice, this means that the cell doors are shut, the interlocks are engaged, all safety signals have been reset, all emergency stop devices are disengaged and the robot has been set up to enable remote control of the joints via ROS 2. If this is not the case, the architecture will only lose the ability to directly control the physical robot, end effectors and actuators. The architecture can still be used to run simulations in Gazebo, the perception system will still be functional, and the architecture can monitor any input and output variables of the sensors, joints and the other hardware of the cell.

The PLC is primarily responsible for the control of the hardware in the cell including the safety, I/O, pneumatics and robot controllers. The safety system states may be monitored by the R3M architecture via the PLC, but the states cannot be changed or reset through the R3M architecture. This must be done physically, such as releasing an emergency stop switch, or via the PLC, such as locking or unlocking an interlock. The PLC can be used to control other peripheral hardware such as valves and solenoids for pneumatic air supply to the cell (used for some end effectors and for parts and fixtures for specific use cases).

Any Input and Output signals directly from the robot I/O or other I/O blocks setup around the cell can be monitored or changed via the PLC or the R3M architecture. This can primarily be used for actuating end effectors or other valves/actuators for fixtures used in specific use cases. I/O signals are also used to trigger camera image acquisition or measurements from other sensors.

Finally, a physical industrial screen is used to monitor the status of the R3M architecture and the signals available from the PLC. A HMI was

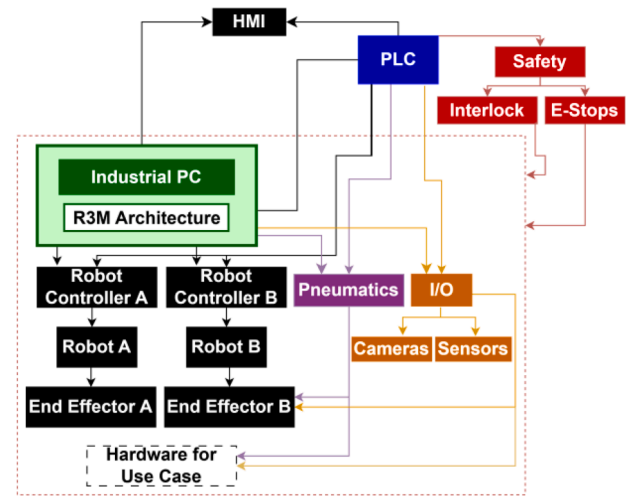


Fig. 4. Hardware overview for an industrial reconfigurable system using the R3M Architecture.

developed as part of the R3M architecture which was used as a graphical user interface with features such as starting a program, monitor the progress of a running program and observing the current robot position. The HMI can also monitor the separate signals coming from the Master PLC such as safety status of E-stops and interlocks and other control parameters.

4.5. Program execution and real-time adaptation

In this section, the integration of the APG and Perception modules into the R3M Cell architecture is introduced, along with an in-depth explanation of their interactions and coordination during the environment launch, management and program execution processes.

4.5.1. System components

The R3M system architecture is composed of three core modules: R3M Cell Orchestrator, R3M APG, and R3M Perception, which work together to enable the execution and real-time adaptation of robot programs. The diagram in Fig. 5 illustrates the architecture and the interactions between these components.

The **R3M Cell Orchestrator** coordinates the full execution environment. It launches and manages the robot cell in simulation or on real hardware, supervises the execution of robot skills, and acts as the central communication point between the robot, APG, and Perception modules. Skill execution is handled through a ROS 2 service interface, allowing the orchestrator to receive requests, invoke the corresponding robot operation, and return execution feedback in a consistent and modular way.

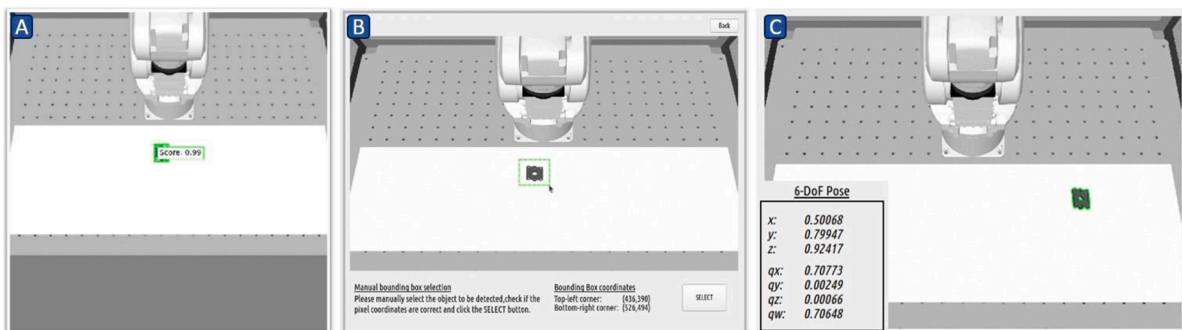


Fig. 3. Manual intervention in the object detection & pose estimation process, using a custom Human-Machine Interface (HMI). [A] One-Shot Detection result. [B] Manual definition of the object using a bounding box. [C] 6 DoF pose estimation result.

The **R3M APG** module enables autonomous program execution by connecting a pre-trained RL agent with the robot cell through the orchestrator. It uses structured recipe definitions and use-case information to select and request robot skills during execution. The orchestrator returns execution feedback after each requested action, allowing the APG module to update the agent state and determine subsequent actions. In this way, APG supports closed-loop task execution and adaptive decision-making based on real-time system feedback.

The **R3M Perception** module provides real-time object detection and 6-DoF pose estimation within the robotic cell, enabling adaptive robot behaviour. By continuously supplying environmental state data to the APG Agent, it ensures that execution decisions reflect current conditions. The module operates as a ROS 2 Service Server that executes a Perception Skill. Upon request, it detects objects with available CAD models and estimates their poses. The resulting object states are published to dedicated ROS 2 topics, with each object represented individually. During execution, the R3M Orchestrator retrieves this data to update skill feedback and forwards it to the APG Agent for ongoing state evaluation. This integration supports dynamic response to environmental changes and improves execution accuracy and robustness.

The behaviour of these modules is configured through three structured .yaml files describing the robot cell, the use case, and the robot skill recipes. The Robot Cell Information file defines the robot and end-effector configuration; the Use-Case Information file defines the product and process context and the associated recipe references; and the Robot Skill Recipe file defines the executable skill logic and its parameters. Together, these files provide the orchestrator with the information required to initialise the execution environment, instantiate the product and process context, and locate the recipes used during runtime. Robot configuration, product information, process logic, and skill-execution definitions are therefore externalised from the core software modules and represented in a reusable, structured form. Representative examples of these configuration files are provided in [Appendix B](#).

Once this configuration data is loaded, the orchestrator launches the execution environment by initialising the robot cell, the planning and control layer, the APG and Perception modules, and the service interfaces used for skill execution. This launch logic is the same whether

the R3M Cell is executed in simulation or on real hardware, reinforcing the shared execution structure of the architecture.

4.5.2. Skill execution process

The R3M Cell Orchestrator manages the execution of robot programs through a dedicated ROS 2 service, `/r3m_SkillExecution`, which acts as the interface between the APG module and the robot cell. During execution, the APG module sends a recipe ID as input, identifying the skill to be executed. The same service can also be manually triggered for testing purposes. Upon receiving a request, the orchestrator retrieves the corresponding recipe, loads the relevant skill parameters, executes the operation, and returns feedback describing the execution result.

The feedback exchanged during execution relies on a set of custom ROS 2 message types representing the robot state, product state, and liaison conditions. The `skill.msg` structure encapsulates execution time, error information, robot state, product states, and liaison information through nested message types including `rob.msg`, `product.msg`, `pose.msg`, and `liaison.msg`. These message structures provide a consistent data model for monitoring execution and exchanging system-state information between the robot cell, the orchestrator, and the APG module. Representative definitions of these message structures are provided in [Appendix B](#).

4.5.3. Program execution using a pre-trained RL agent in MATLAB

This section describes how a pre-trained RL agent is integrated into the R3M APG module and used to drive closed-loop program execution within the R3M system. A concrete implementation of this generic execution logic is later illustrated in a specific use-case demonstration in [Section 5.4](#), where [Fig. 12](#) provides an example of the closed-loop information flow between the orchestrator, the APG module, the Perception module, and the robot cell.

Once trained, the RL agent is exported from MATLAB/Simulink as an executable Python program and integrated into the R3M-APG module. At runtime, the agent receives the current system state as input and returns an action corresponding to a recipe ID. This recipe ID is sent to the R3M Cell Orchestrator, which interprets it as a request to execute a specific skill.

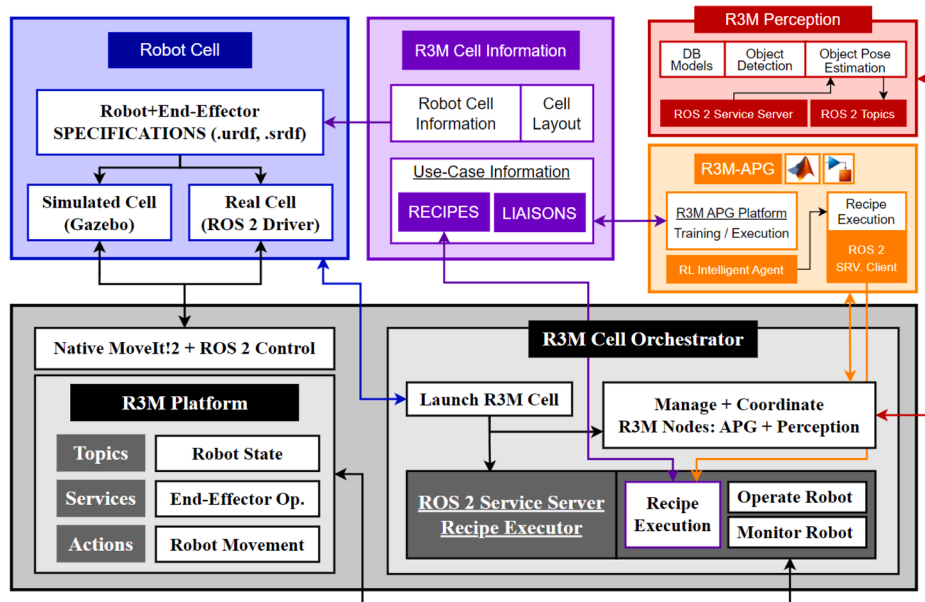


Fig. 5. R3M system architecture showing the integration of APG and Perception within the R3M Cell. The figure extends the generic R3M Cell architecture by illustrating how the Cell Orchestrator coordinates robot operation, recipe execution, and real-time state updates across the APG and Perception modules. The APG module selects and requests recipes through the orchestrator, while the Perception module provides object detection and pose estimation services whose outputs are published and reused during execution. Together, these connections show how robot control, product/process information, perception feedback, and learning-based decision-making are combined within a unified execution framework.

Before execution, the orchestrator queries the R3M Cell and Use-Case Information database to retrieve the parameters associated with the selected recipe. Depending on the requested skill, execution is then delegated either to the robot cell or to the perception module. Robot skills trigger actions such as motion execution or end-effector operation, while perception skills trigger object detection and pose estimation. In both cases, the corresponding module returns updated system-state information to the orchestrator. The orchestrator then assembles execution feedback, including equipment state, product state, and process outcome, and returns it to the APG module. This updated state becomes the next input to the RL agent, closing the execution loop. The process continues iteratively until the required liaison conditions are satisfied, indicating that the task has been completed successfully.

A key strength of the R3M architecture is its modularity. For example, the perception component used in this implementation can be replaced by alternative perception pipelines without major changes to the rest of the architecture. This makes the APG methodology scalable and adaptable across different industrial applications.

The robustness of this execution process is grounded in liaison validation. In R3M, liaisons represent the target relational conditions between the objects involved in a use case; that is, the expected final relative poses or assembly relationships that define successful task completion. A liaison includes the parent and child objects, the target transformation between them, and the admissible error thresholds used for validation. During execution, these liaisons are stored as part of the use-case information and represent the objective states that the system aims to achieve. The orchestrator evaluates them against the object-pose estimates returned by the perception skill during execution to determine whether the intended product state has been reached.

Rather than validating every atomic action individually, the system therefore evaluates whether the expected liaisons between parts are satisfied at the end of a task. If the required liaisons are not achieved, the updated product state is returned through the perception feedback, and the RL agent generates a new action to reattempt the task. This enables the system to recover from execution errors and continue toward successful task completion without external intervention.

4.6. Component integration

Fig. 6 presents the runtime integration architecture of the R3M cell. The R3M Cell Orchestrator is the central coordination component, consuming services and data from the surrounding subsystems through typed interfaces. Industrial safety and I/O are encapsulated in the PLC/Safety & I/O component, which provides ISafetyIO to the Orchestrator for interlocks, mode control, and emergency-stop handshakes. The PLC also provides a IStatusFeed that the HMI requires for operator supervision and fault acknowledgement. Motion authority is delegated to R3M Robotics Control, which provides IMotionControl; the Orchestrator requires this interface to issue trajectories and monitor execution. Task knowledge originates in R3M APG, which provides ISkillExecution for skill invocation and provides IRecipeLookup to Robotics Control for recipe parameterisation. The R3M Database provides IRecipeRepository, which APG requires to retrieve and persist product, process, and skill assets. Perceptual grounding is delivered by R3M Perception, which provides IObjectState to the Orchestrator, enabling state estimation, verification of preconditions, and recovery logic. Lollipop and socket symbols denote provided and required interfaces, respectively, emphasising contract-based coupling and separation of concerns. Collectively, the diagram formalises data and control pathways for safety, motion, perception, and task synthesis, yielding a reproducible and auditable integration pattern suitable for adaptive manufacturing.

5. Use case demonstration: AI-enhanced object manipulation

The integration of the R3M framework was validated through two industrially relevant use cases, namely Cube Kitting and Cylinder Stacking, carried out at Cranfield University and the Advanced Manufacturing Research Centre (AMRC). These demonstrators were executed on the ABB IRB120 and UR3 at Cranfield, and on the ABB IRB1200 at the AMRC, corresponding to Cube Kitting and Cylinder Stacking, respectively. To ensure consistency, the APG modules (Sections 5.1 and 5.2) and the agent execution process (Section 5.4) were applied within the Cylinder Stacking scenario, while the perception pipeline developed in University of Sheffield (Section 5.3) was validated across both Cube Kitting and Cylinder Stacking. At the AMRC

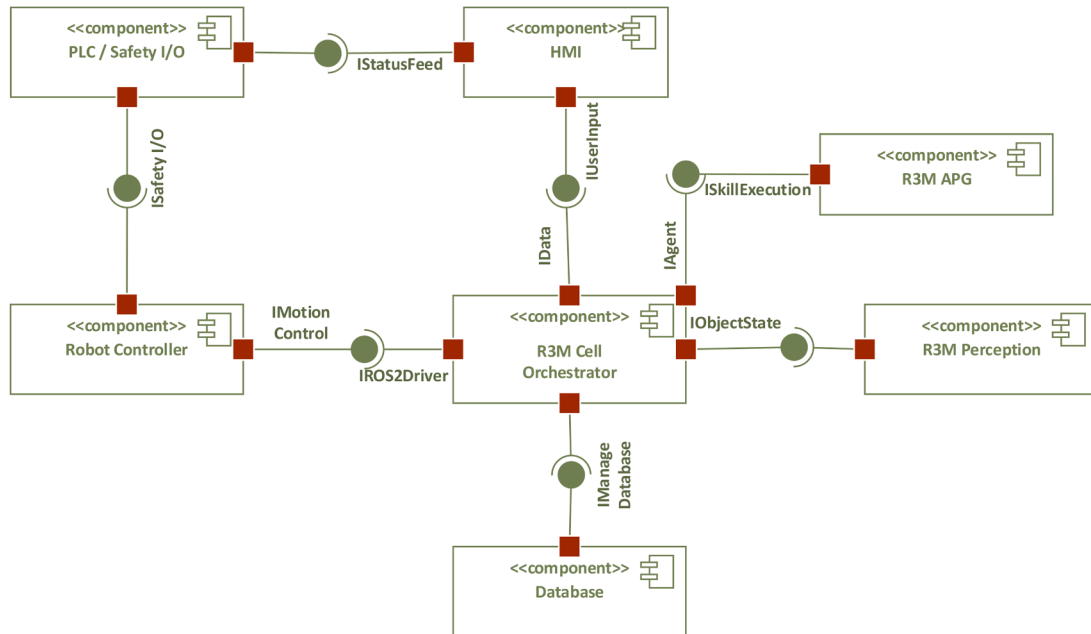


Fig. 6. Component level integration architecture of the R3M Cell, centred on the Orchestrator as the coordinating element for robotics control, PLC safety and I/O, APG, perception, database services, and HMI supervision. The provided and required interfaces define the control, data, and feedback pathways needed for safe execution, task synthesis, perception driven state estimation, and operator oversight.

(Section 5.4), complete integration was demonstrated through both use cases, thereby confirming the scalability of the framework across sites and robotic platforms.

5.1. Defining functional requirements for the cylinder stacking use case

The effectiveness of the APG methodology introduced by the R3M project depends fundamentally on the completeness and coherence of the information provided during the requirements definition phase.

For the *Cylinder Stacking* use case, this phase involves capturing and formalising all relevant data across three interrelated domains: product, process, and equipment. This structured knowledge forms the backbone of subsequent APG stages and is represented using AML, a standard well-suited for describing manufacturing systems with semantic precision and interoperability.

5.1.1. Product domain specification

The product defined for this use case comprises two individual components: a blue cylinder (bottom) and a green cylinder (top). The required task is to stack the green cylinder directly on top of the blue cylinder. To represent this behaviourally, the product model includes two skill requirements, each described with the common attribute Skill Type set to "pick-and-place task". These skill requirements represent the fundamental actions required to manipulate and position the parts.

The product model also includes precedence constraints between these skill requirements to maintain the correct sequence of operations, first picking and placing the bottom cylinder, followed by placing the green cylinder on top of the blue one. Furthermore, the liaison definitions specify how the two cylinders interact physically to form the assembled product. In this model, two liaisons are defined abstractly, indicating the spatial and relational configuration necessary to complete the stacking task.

5.1.2. Equipment domain specification

The physical system used for this use case includes three main equipment modules: a 6-DOF industrial robot arm, a pneumatic gripper, and an overhead vision camera. Each module exposes its capabilities through well-defined atomic skills, each having distinct functional roles:

- The robot module supports a move skill, which is essential for translating the robot end-effector between 6DoF robot poses. This skill is parameterised by pose and speed attributes.
- The gripper module provides hold and release skills, allowing it to grasp and release objects. These are typically parameterised using binary settings (e.g., open/close).
- The camera module supports inspect and detect skills, which can aid in object recognition and positioning tasks.

Each atomic skill in the equipment domain is defined with explicit input and output parameters. These parameters are crucial for enabling dynamic recipe generation. Moreover, skills can report KPIs such as execution time, power usage, or material consumption through dedicated information ports, which are later used in the reinforcement learning phase to optimise performance.

5.1.3. Process domain specification

The process model for the *Cylinder Stacking* task is represented using Composite Recipes. Fig. 7 highlights these structured templates encapsulating the domain knowledge about how tasks should be logically and operationally decomposed. Three primary composite recipes are defined for this use case:

- Pick and Place – a higher-level abstraction that encompasses object manipulation routines.
- Pick – a more granular recipe focusing on grasping the object.
- Place – a recipe for positioning and releasing the object.

Each composite recipe includes its own skill requirements, also tagged with Skill Type, ensuring consistent traceability and automatic matching during the recipe generation phase. The nesting and referencing of composite recipes provide a scalable way to handle process complexity while supporting reusability across different tasks.

5.1.4. Unified model representation

All product, process, and equipment descriptions are encoded in AML, which allows for semantic linkage and structured representation of interdependencies across domains. The Skill Type attribute, consistently applied across skill requirements, composite recipes, and equipment skills, acts as the key matching parameter for the APG methodology. This attribute essentially serves as the de facto standard that bridges domain-specific representations, allowing the automation system to interpret, deconstruct, and reassemble knowledge components for program generation.

5.2. APG methodology implementation for the cylinder stacking use case

Building upon the structured domain models defined during the requirements definition phase, the APG methodology is applied to realise the *Cylinder Stacking* use case. The APG implementation in R3M follows a three-step process: Recipe Generation, Reinforcement Learning, and Code Generation. This section details the execution of these steps, emphasising how the semantic alignment between product, process, and equipment models enables end-to-end automation.

5.2.1. Step 1: recipe generation

The recipe generation phase begins with automated skill matching, wherein the system attempts to fulfil the skill requirements of the product using available equipment capabilities. In the *Cylinder Stacking* scenario, the two product-defined skill requirements each characterised by the Skill Type "pick-and-place task" are examined for direct matches with the equipment's atomic skills. However, as expected, atomic skills like move, hold, and release are too granular to directly fulfil the high-level product requirements.

To resolve this, the system turns to the library of Composite Recipes, which bridge the abstraction gap between product needs and low-level equipment functions. The system identifies that the "Pick and Place" composite recipe matches the skill type required by the product. However, this recipe itself contains further skill requirements such as pick and place that again need to be resolved.

The system recursively searches the composite recipe library and identifies individual recipes for pick and place, which in turn decompose into skill requirements that match directly with equipment-level atomic skills. Specifically, the pick recipe maps to a sequence of move and hold skills, while the place recipe corresponds to move and release skills. At this point, all skill requirements can be grounded in executable skills available in the equipment domain.

Each equipment skill is associated with parameters. For example, the move skill of the robot requires pose and speed inputs. To accommodate variability in real-world conditions, the system considers a range of possible parameter settings. For the stacking task, five pre-evaluated robot poses (representing viable pick-and-place positions) are defined based on expert knowledge. These preset positions form the basis for generating a diverse set of skill recipes with parameter values which will later be evaluated for optimality.

5.2.2. Step 2: reinforcement learning

For the *Cylinder Stacking* use case, the reinforcement learning (RL) problem is formulated to optimise the selection of parametrised recipes that govern the pick-and-place operations required to assemble two cylinders. The objective of the learning process is to minimise the total execution time of the task by selecting the most efficient sequence of actions from a set of alternative recipes generated during the recipe generation phase.

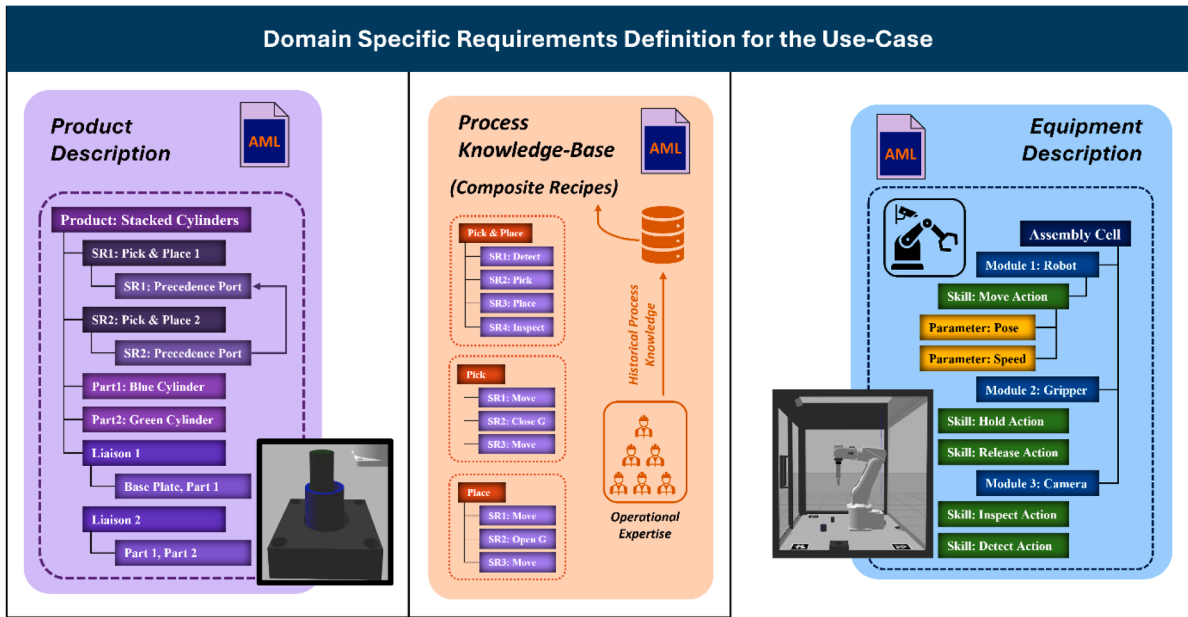


Fig. 7. Domain specific requirements model for the Cylinder Stacking use case. The representation captures the product, process, and equipment information required for automated programme generation, including skill requirements, composite recipes, skill types, and executable equipment skills that support the generation of feasible stacking sequences.

The learning environment is implemented using a digital twin of the robotic cell, where the system state captures the progress of the stacking task at each decision step. The state representation includes the poses of the two cylinders, the status of the two liaisons (i.e., the placement of the first cylinder on the base and the stacking of the second cylinder on top of the first), as well as the pose of the robot and the state of the gripper. Based on this state, the RL agent selects an action corresponding to a recipe ID, where each recipe represents a parametrised skill instance such as pick, move, or place.

A key aspect of this use case is the introduction of multiple alternative recipes for each robot pose. Specifically, for every movement involved in the pick-and-place operations, five alternative recipes are defined. These alternatives correspond to different feasible robot poses that achieve the same task objective, such as approaching an object from different orientations or executing a placement from slightly varied configurations. This significantly increases the size of the action space, as the agent must not only determine the correct sequence of operations but also identify the most efficient pose configuration among the available alternatives. Consequently, the RL problem becomes a combinatorial optimisation task over both sequence and parameter selection.

The reward function is defined as a penalty-based measure of execution time, such that $r_t = (-T(a_t))$, where $T(a_t)$ denotes the execution time of the selected recipe. Under this formulation, the agent is encouraged to minimise execution time by selecting faster and more efficient recipes. The learning process is carried out using the Proximal Policy Optimization (PPO) algorithm, which iteratively updates the policy based on interactions with the simulated environment.

The training progress for the Cylinder Stacking use case is illustrated in Fig. 8. The plot shows the episode reward, its 5-episode moving average, and the associated confidence band. In the initial phase of training, the reward values exhibit significant variability, reflecting the exploratory behaviour of the agent as it evaluates different combinations of recipes. This variability is particularly high due to the presence of multiple alternative poses for each action, which introduces many possible configurations. As training progresses, the agent begins to favour more efficient recipes, resulting in a steady increase in reward values (i.e., reduction in execution time penalties).

A clear convergence trend is observed as the training advances

beyond approximately 2000 episodes. The moving average gradually approaches the final best reward, while the confidence band narrows, indicating reduced variability in action selection. Importantly, the final converged reward represents the best performance achieved during training, and no intermediate episode exceeds this value. This confirms that the agent has successfully identified the optimal combination of recipes for the stacking task. The stabilisation of the reward in the final episodes further indicates that the policy has converged and consistently produces near-optimal decisions.

Overall, the results demonstrate that the RL setup is capable of effectively handling the increased complexity introduced by multiple alternative recipes. The agent not only learns the correct sequence of actions required for the stacking task but also optimises the selection of robot poses to minimise execution time. This highlights the ability of the APG methodology to address both sequencing and parameterisation challenges in automated robot programming.

5.2.3. Step 3: code generation

In the final stage of the APG methodology, the trained reinforcement learning agent for the Cylinder Stacking use case is converted into an executable program that can be deployed within the robotic system. The agent, which has been trained using Proximal Policy Optimization (PPO) in MATLAB, is exported using MATLAB's code generation capabilities and packaged as a Python module through the MATLAB Compiler SDK. This conversion enables the learned policy to operate independently of the MATLAB environment, while execution on a Linux-based system is supported through the MATLAB Runtime, allowing royalty-free deployment.

The resulting Python module implements the learned policy as a function that maps the current system state to an action. For the Cylinder Stacking task, the system state includes the poses of the two cylinders, the status of the two liaisons (base-to-cylinder and cylinder-to-cylinder), the robot pose, and the gripper state. At runtime, this state is continuously updated based on feedback from the robotic cell or its digital twin. Upon receiving a state input, the Python-based agent evaluates the current progress of the stacking task and outputs an action in the form of a recipe ID, corresponding to a parametrised skill such as pick, move, or place.

The generated action is passed to the R3M Cell Orchestrator, which

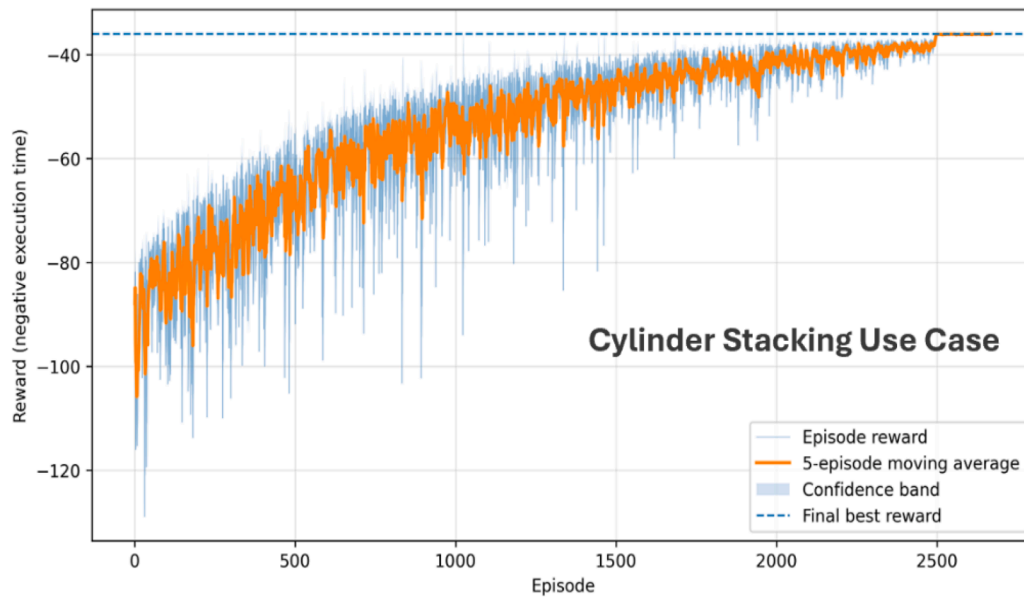


Fig. 8. Reinforcement learning training performance for the Cylinder Stacking use case. The reward trajectory, moving average, and confidence band characterise the agent's progression from exploratory recipe selection towards more efficient policies, demonstrating convergence towards reduced execution time penalties across training episodes.

interprets the recipe ID as a request to execute a specific skill instance. The orchestrator retrieves the corresponding recipe parameters, including target poses, motion constraints, and gripper commands, from the system database. Based on the type of skill, the execution request is then directed either to the robot control module or to the perception module. For example, if the selected action corresponds to a pick operation, the robot is commanded to move to the detected position of the cylinder, close the gripper, and update its internal state accordingly. Similarly, placement actions are executed by moving the cylinder to the target stacking position while ensuring alignment with the required liaison.

A key aspect of this execution process is the continuous feedback loop between the robotic system and the trained agent. After each action is executed, the updated system state includes the new poses of the cylinders, the status of the liaisons, and the robot cell's status is returned to the agent. This updated state reflects the progress of the stacking task and serves as the input for the next decision step. If a desired liaison is not achieved, for example due to misalignment or unsuccessful placement, the state will indicate that the assembly condition has not been satisfied. In response, the agent selects an alternative recipe to reattempt the operation, such as adjusting the approach pose or repeating the pick-and-place sequence. This enables the system to adapt dynamically to execution uncertainties.

The execution process continues iteratively until both liaisons are satisfied, indicating that the first cylinder is correctly placed on the base and the second cylinder is successfully stacked on top of the first. At this point, the system reaches the terminal state of the task. The resulting program therefore consists of a closed-loop decision-making structure, where the trained agent continuously selects optimised actions based on the real-time state of the robotic cell. This ensures that the final execution sequence is not a fixed script, but a dynamically generated program that adapts to the current situation while consistently aiming to minimise execution time.

5.3. Object detection and pose estimation via perception systems

In the *Cube Kitting* and *Cylinder Stacking* tasks, the one-shot object detection and 6DoF pose estimation presented in Section 4.3 are

performed sequentially to facilitate precise manipulation. Initially, all relevant objects are detected simultaneously using a One-Shot Detection approach, guided by query images. The detection results, as illustrated on the top-left side of Fig. 9, include the bounding boxes, object IDs, and confidence scores ranging from 0 to 1.

Following the detection step, the identified objects are processed by a 6DoF pose estimator to determine their full six-degree-of-freedom (6DoF) poses. The estimated poses are visually represented by green contour lines outlining the objects, as shown on the right side of Fig. 9. This integrated pipeline enables accurate object localization and pose estimation, which are crucial for downstream manipulation tasks in robotic applications.

Initially, the system performs an initial inference using a 6DoF estimator. After obtaining the initial pose estimation, the perception system switches to a faster 6DoF estimator that enables real-time pose estimation. These real-time 6DoF poses are continuously published through a ROS 2 Topics, with each object assigned a dedicated topic, this being identified with the object's name. The robot program execution module subscribes to these topics as needed, ensuring seamless access to up-to-date pose data for precise manipulation tasks. Since the perception system runs in the background with minimal resource consumption, it does not need to be explicitly invoked each time perception information is required. This continuous real-time publishing mechanism significantly enhances the efficiency and accuracy of object handling in robotic applications, such as pick-and-place tasks.

The perception system follows a structured pipeline, as illustrated in Fig. 10. To facilitate pose estimation, four image queries and four corresponding mesh files are pre-prepared, each representing one of the four cubes. The image queries capture both the colour and geometric features of the target objects, enabling the system to distinguish between cubes of identical shape but different colours. Since the 6DoF pose estimator operates only within the regions identified by the One-Shot Detection results, texture information is not required. This selective focus improves computational efficiency by limiting the estimation process to the relevant object regions, ensuring accurate pose predictions for subsequent robotic manipulation.

The perception system follows a structured pipeline, as illustrated in Fig. 10. To support pose estimation, four image queries and four

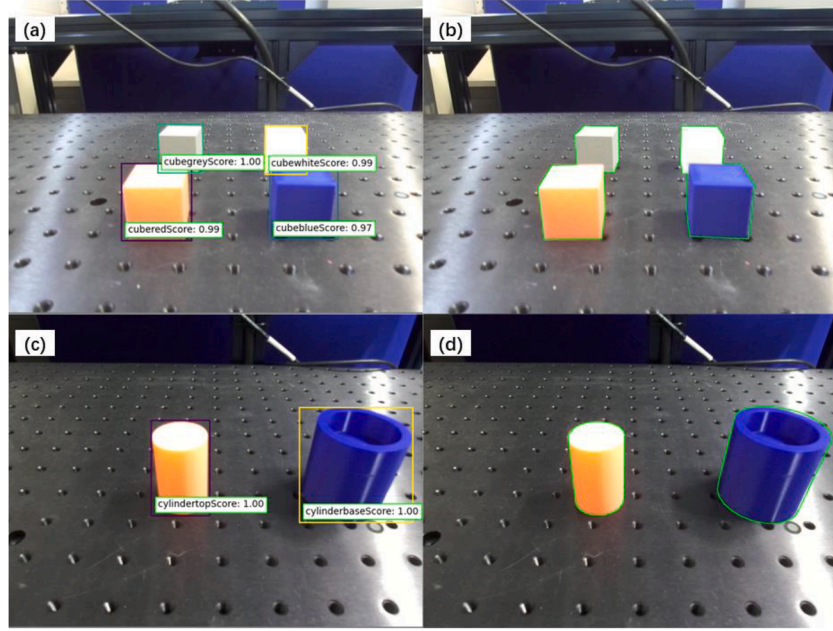


Fig. 9. (a) One-Shot Detection results for Cube Kitting. (b) 6DoF pose estimation results for Cube Kitting. (c) One-Shot Detection results for Cylinder Stacking. (d) 6DoF pose estimation results for Cylinder Stacking.

corresponding mesh models are prepared, each representing one of the four cubes. The image queries provide colour and geometric cues, enabling the system to distinguish between cubes with identical geometry but different colours. The 6DoF pose estimator is then applied only to the object regions identified by the one-shot detection module. Within these regions, pose estimation primarily relies on the geometric information provided by the mesh models rather than texture features. This region-constrained estimation improves computational efficiency by limiting pose inference to relevant object areas.

5.4. Execution and validation of the APG-generated program for the cylinder stacking use case

This section presents the integrated validation of the R3M framework through the Cube Kitting and Cylinder Stacking use cases. Both demonstrations were carried out in the RECON Cell at the AMRC's Factory 2050 facility using an ABB IRB 1200–5/0.9 robot equipped with a pneumatic Zimmer two-jaw parallel gripper. The purpose of this section is to demonstrate how the R3M architecture supports complete program execution, perception-informed adaptation, and validation of task

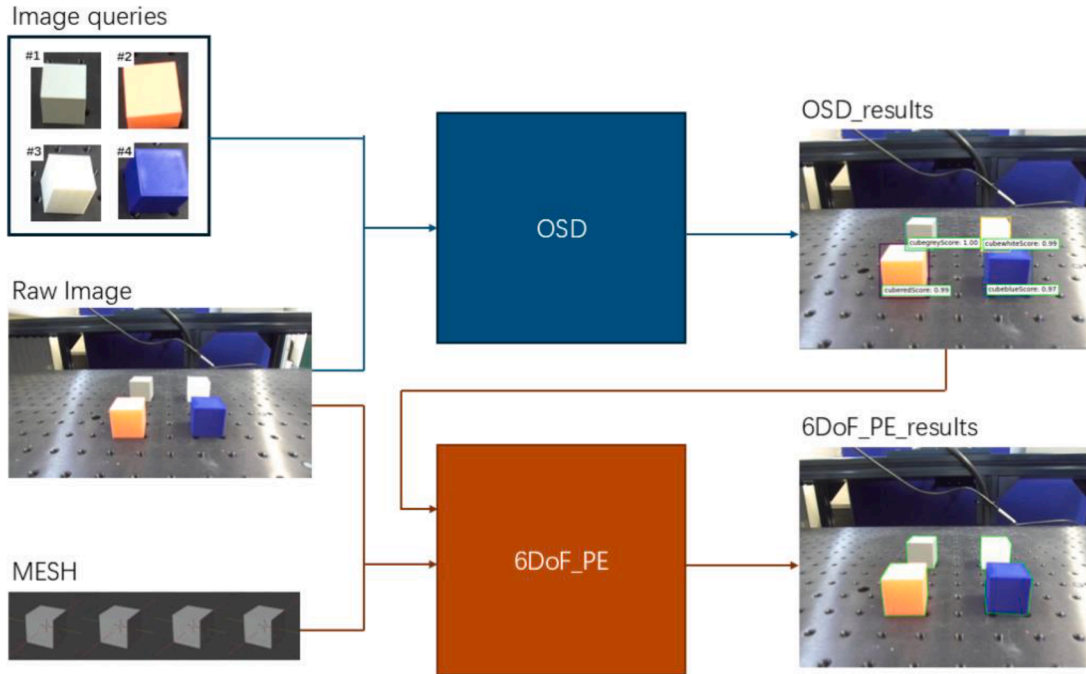


Fig. 10. Perception inference workflow for the Cube Kitting use case, linking query image preparation, mesh model selection, one shot object detection, region localisation, and six degree of freedom pose estimation. This process enables pose recovery for task relevant cube objects using CAD informed perception rather than object specific retraining.

completion in representative industrial scenarios.

Although the perception methodology described earlier in the paper is primarily based on the OSD+M6D pipeline, the integrated demonstrations reported in this section employed YOLOv11 within the R3M Perception module. This choice was motivated by practical runtime considerations, as YOLOv11 offered substantially faster loading and execution, enabling more immediate response during perception-skill execution and smoother overall operation in the demonstration environment. This implementation choice does not alter the surrounding architecture; rather, it highlights an intended property of the framework, namely that the perception layer is modular and allows alternative pipelines to be integrated and evaluated without structural changes to the rest of the R3M system.

5.4.1. Cylinder stacking use case

For the Cylinder Stacking use case, two cylinders and a tray were placed between four ArUco markers inside the RECON Cell. Custom 3D-printed fingertips were mounted on the gripper, with a two-step geometry that allowed stable grasping of both the thinner and wider cylinders. After successful completion of the task, the larger cylinder was placed inside the tray and the smaller cylinder was stacked inside it. The use case was tested with different initial positions of both cylinders and the tray, demonstrating the ability of the framework to handle positional variation while maintaining correct execution.

5.4.2. Cube kitting use case

For the Cube Kitting use case (Fig. 11), an external camera mounted on a tripod was used to detect the cubes and tray, again positioned between four ArUco markers. Custom metal fingertips were fitted to the gripper to allow top-down access to the cubes while avoiding collision with adjacent parts during placement. After successful execution, all four cubes were placed within the tray according to the required final arrangement. The task was demonstrated with cubes placed in different initial locations, including partially overlapping configurations, showing that the framework could adapt to varying spatial setups through the perception and execution pipeline.

5.4.3. Use-case implementation and execution of the R3M architecture

The execution and validation of APG-generated programs within the R3M framework are illustrated through the Cylinder Stacking use case. As shown in Fig. 12, this use case provides a concrete implementation of the generic closed-loop execution logic introduced in Section 4.5. In this scenario, the APG module controls the sequential stacking of two

cylinders by issuing recipe requests to the orchestrator, which then coordinates robot-skill execution, perception updates, and liaison validation.

The execution starts with the APG agent selecting a recipe ID associated with one of the required operations in the task. For the Cylinder Stacking case, these recipes correspond to atomic robot skills such as PTP movements to home, approach, or transfer poses, LIN movements to pick and place poses, and gripper actions for closing and opening the Zimmer gripper. As illustrated on the right-hand side of Fig. 12, these skills are parameterised by type, speed, pose, and action values. During runtime, the orchestrator retrieves the corresponding recipe specifications and dispatches the request either to the robot cell or to the perception module, depending on whether the next step requires physical execution or an updated estimate of the product state.

The Cylinder Stacking process naturally unfolds in two sub-sequences: first, the pick-and-place of the bottom cylinder onto the workspace, and second, the pick-and-place of the top cylinder into the bottom cylinder. Each sub-sequence combines robot motions, gripper operations, and intermediate perception updates. In normal execution, this results in a sequence of approach, pick, transfer, and placement operations selected by the RL agent to complete the stacking task efficiently.

Task completion is not determined only by executing these robot skills, but by validating the required liaisons shown in Fig. 12. In this use case, two liaisons define success: the first checks that the bottom cylinder has been placed correctly on the workspace, and the second checks that the top cylinder has been placed correctly relative to the bottom cylinder. Each liaison encodes a target spatial relationship, including parent and child objects, the expected relative transform, and admissible error bounds. During execution, the orchestrator compares these target conditions against the object poses estimated at runtime by the perception module. Only when both liaisons are satisfied is the stacking task marked as complete.

This validation mechanism is what gives the process its robustness. If, for example, a cylinder is misplaced or slips during placement, the corresponding liaison is not satisfied even if the robot skill itself has been executed. In that case, the failure state is returned to the APG agent, which generates a new action sequence to recover and continue execution. The result is a closed-loop assembly process in which the Cylinder Stacking task is not only executed, but also verified and corrected automatically through the same R3M architecture.

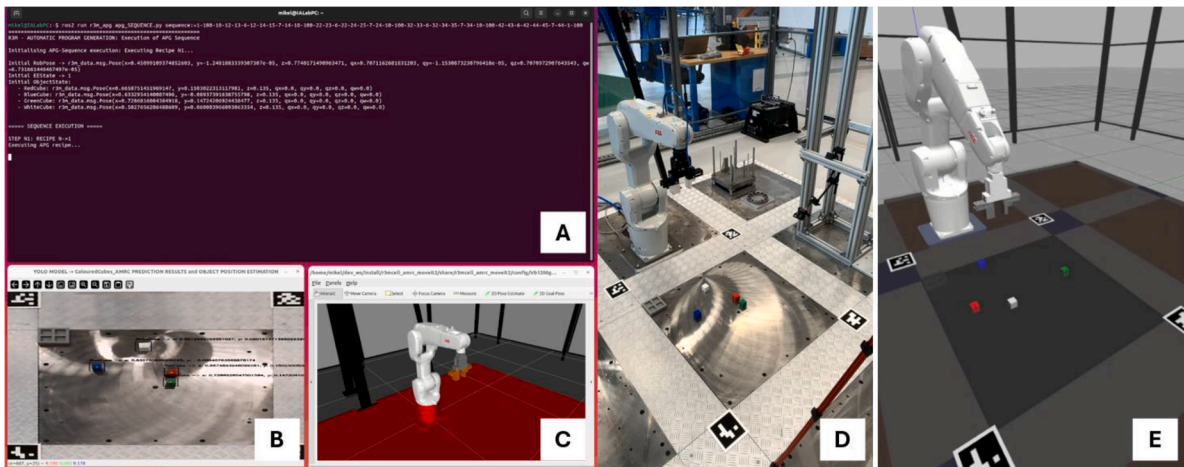


Fig. 11. Experimental and digital views of the RECON Cell used for integrated R3M demonstrations. A shows the terminal output of the execution process, B shows the output of the YOLOv11-based object detection and pose estimation within the R3M Perception module, C shows the robot digital twin in RViz, D shows the real robot and camera view in the physical cell, and E shows the corresponding simulated robot in Gazebo Sim. Together, these views illustrate the simultaneous operation of execution, perception, visualisation, and simulation within the R3M architecture.

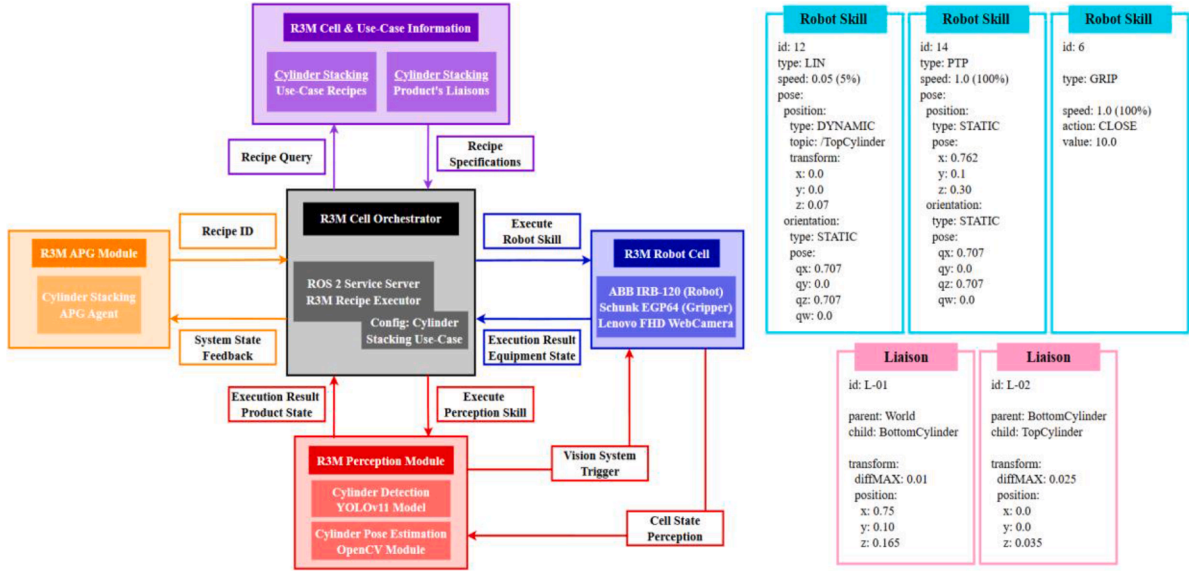


Fig. 12. Closed-loop information flow in the R3M Cell for the Cylinder Stacking use case. The figure shows a concrete implementation of the generic execution logic introduced in Section 4.5, illustrating how the APG module, the R3M Cell Orchestrator, the Perception module, and the robot cell exchange recipe requests, skill results, and system-state feedback during runtime. The right-hand side also shows examples of robot-skill definitions and liaison conditions used to validate successful task completion.

6. Results & discussion

The demonstrations presented in Section 5 validate the ability of the R3M framework to integrate APG, perception, and agent execution across different industrial scenarios, robotic platforms, and research sites. These results show that the framework can support scalable and adaptable deployment while linking high-level domain models with execution-level mechanisms for flexible task execution in heterogeneous environments. Building on these findings, Section 6 provides a critical reflection on these outcomes, discussing the extent to which the proposed objectives have been achieved, together with the main strengths and limitations observed during implementation.

6.1. R3M cell: software architecture

The R3M Cell architecture was evaluated in terms of reliability, consistency, and reconfigurability across multiple robotic platforms and tasks. Performance metrics included environment launch time, skill execution responsiveness, and overall program robustness in both simulation and real-world deployments. Particular emphasis was placed on sim-to-real consistency, assessing the transferability of results from Gazebo simulations to physical systems. All experiments were executed through automated Python-based test pipelines leveraging ROS 2 communication interfaces, enabling large-scale, reproducible evaluation across diverse scenarios.

6.1.1. Test design

All tests were executed through automated Python scripts that managed the launch and shutdown of the ROS 2 environment, issued commands through ROS 2 Topics, Services, and Actions, and recorded the results in structured .csv files for subsequent analysis. This enabled large-scale, reproducible evaluation across multiple robots, environments, and use cases with minimal human intervention.

Three categories of tests were conducted to evaluate the R3M Cell architecture:

6.1.1.1. ROS 2 environment launch. These tests measured launch time and launch success across simulation and real-robot environments. Three subtests were performed:

- Gazebo Simulation Launch: 1700 iterations across 17 different R3M robot-cell environments, each repeated 100 times, measuring robot-spawn and environment-launch behaviour.
- Gazebo + MoveIt!2 Launch: 1700 iterations across the same 17 environments, measuring robot spawn, controller loading, robot-operation node initialisation, and overall environment launch time.
- Real Robot Control + MoveIt!2 Launch: 1000 iterations split equally between the ABB IRB120 and UR3 cells at Cranfield University, measuring robot connection, ROS 2 driver initialisation, controller loading, monitoring, and overall launch performance.

6.1.1.2. R3M skill execution. These tests evaluated the reliability and consistency of a standard three-skill execution sequence composed of joint-space motion, Cartesian motion, and return-to-home. Each test iteration involved launching the environment, executing the three-skill sequence, and closing the environment. The test campaign comprised:

- Simulation Tests: 4000 iterations distributed across four simulated R3M cells (IRB120 and UR3 at Cranfield, IRB1200 and IRB6640 at AMRC-Sheffield).
- Real Robot Tests: 2000 iterations across the ABB IRB120 and UR3 cells at Cranfield.

For each iteration, launch success, APG node initialisation, skill-execution success, robot-motion execution time, and total sequence execution time were recorded.

6.1.1.3. R3M program execution. To assess the architecture under a more complex task, the Cube Kitting program was used as a program-level benchmark. The program consisted of 41 skills in simulation, including perception steps, and 37 skills on real hardware. It therefore evaluated not only atomic skill reliability, but also end-to-end execution robustness, including liaison validation. The test campaign comprised:

- Simulation Tests: 2000 iterations in total (1000 each on the IRB120 and UR3 cells at Cranfield), with automated success verification based on Gazebo object-pose checks.
- Real Robot Tests: 200 iterations in total (100 each on the IRB120 and UR3 cells at Cranfield), where cube-placement verification was

performed manually while all other data collection remained automated.

6.1.2. Test results analysis

6.1.2.1. Gazebo simulation and R3M environment launch. The launch results show that the R3M Cell architecture achieved very high reliability across both simulation and real-robot environments. In pure Gazebo simulation (Table 4), both robot spawn and overall ROS 2 environment initialisation reached 100.00% success over 1700 iterations, with very low dispersion. The same trend is visible in the full simulated R3M environment (Table 5), where overall launch reached 99.93% success across 6700 iterations, while the APG ROS 2 node reached 100.00% success over 5000 iterations with a mean launch time of 0.54 ± 0.04 s. These results indicate that the architecture can be launched repeatedly and consistently across a large number of simulated configurations.

For the real-robot configurations, both ABB and UR environments also showed high launch reliability, but with different timing behaviour. The ABB environment (Table 6) achieved 99.69% success overall, although several components exhibited larger variability due to occasional outliers in robot connection and driver initialisation. The UR environment (Table 7) achieved 99.88% success, with a slower but more stable overall launch time (11.00 ± 0.28 s, compared with 7.71 ± 1.62 s for ABB). The APG ROS 2 node showed identical launch behaviour in both configurations (0.78 ± 0.08 s), indicating that this component behaved consistently across robot platforms.

6.1.2.2. R3M skill execution. The skill-execution results show that the R3M execution layer was fully reliable for both robot platforms. The standard three-skill test sequence (consisting of joint-space motion, Cartesian motion, and return-to-home) achieved 100.00% success in both simulation and real hardware for the ABB IRB-120 and UR3 robots. This confirms that the ROS 2-based execution pipeline remains stable across both virtual and physical deployments.

In terms of timing, the ABB robot (Table 8) showed highly stable simulation performance (4.79 ± 0.01 s) and fully reliable real execution, although with greater variability (4.00 ± 0.66 s) than in simulation. The UR3 (Table 9) remained highly consistent in both cases, with mean sequence times of 13.14 ± 0.01 s in simulation and 13.28 ± 0.01 s on real hardware. Across both robots, the difference between robot motion time and total skill-execution time remained small (less than 0.30 s per skill), indicating that the R3M framework introduces only a limited and predictable overhead for communication, feedback handling, and execution confirmation.

6.1.2.3. R3M use-case execution. The program-level results in Tables 10 and 11 show that the Cube Kitting use case was executed with high reliability in both simulation and real hardware. In simulation, overall execution success reached 97.40% for the ABB robot and 97.10% for the UR robot, while real-robot trials achieved 100.00% success in both cases. Liaison success followed the same pattern, with 96.99% and 96.70% in simulation, and 100.00% in real execution. These results show that the architecture is capable of executing full task sequences while also validating the intended final object relationships.

Table 4

Gazebo simulation launch results, reporting success rate and timing statistics for robot spawn and overall ROS 2 environment initialisation across 1700 iterations.

Element	Iterations	Success Rate	Mean (s)	Median (s)	STD (s)	Min/Max (s)
Robot Spawn	1700	100.00%	1.89	1.88	0.07	0.58
ROS 2 Env. Launch	1700	100.00%	2.79	2.84	0.14	0.84

The slightly lower success rates observed in simulation were associated with a small number of launch and skill-execution timeout events, rather than systematic failures in task logic or liaison validation. Although the exact cause cannot be identified with certainty, these events are most plausibly attributable to transient timing or computational-load effects during large-scale repeated simulation trials. The execution-time differences between simulation and real hardware were also consistent with the characteristics of the two robot platforms: for the ABB cell, Gazebo execution was slightly slower because of delay introduced when attaching cubes to the end-effector, whereas for the UR3 the real-robot trials were slower because the collaborative robot was operated at reduced speed for safety.

6.1.3. Conclusion

The experimental evaluation demonstrates that the R3M ROS 2-based architecture is both reliable and consistent across simulation and real-robot deployments. Environment launch, skill execution, and full use-case execution all achieved high success rates, with performance variability remaining modest in most cases. Both the ABB IRB-120 and UR3 robots showed 100.00% reliability in the standard three-skill execution sequence, while the Cube Kitting use case consistently achieved execution success rates above 97% in simulation and 100.00% in real trials. These results confirm that the architecture provides a stable and predictable framework for robotic operation, effectively bridging the sim-to-real gap and enabling repeatable deployment across heterogeneous platforms. The modest overhead observed between robot movement execution time and total skill execution time (approximately 0.25–0.30 s per skill) further supports the efficiency of the communication and control pipeline, ensuring responsiveness without compromising stability.

The few errors observed during testing were traced to specific sources, including Gazebo simulation launch timeouts, TCP/IP connection losses or timeouts during robot connection, MoveIt!2 execution errors, and skill-execution timeouts with no feedback returned. These failures were rare, but they highlight clear directions for future refinement. In particular, the lower reliability and slower connection behaviour observed when interfacing with the ABB robot suggest that there is room for improvement in the design and implementation of its ROS 2 driver. Potential solutions include more robust retry mechanisms for launch and connection processes, improved network stability and watchdog monitoring for robot communication, and extended feedback timeout handling during skill execution. In addition, strengthening MoveIt!2 error-recovery routines could further reduce planning and execution failures. Addressing these issues would improve the resilience of the R3M framework and further strengthen its suitability for industrial deployment.

6.2. R3M APG: program generation and execution

The proposed Automated Program Generation (APG) methodology was evaluated across multiple robotic assembly use cases, including LamSheet Pick and Place, Cube Kitting, Cylinder Stacking, Can Pick and Place, and Panel Handling. These use cases were selected to represent a spectrum of task complexities, ranging from simple pick-and-place operations to multi-step assembly scenarios involving multiple parts and liaisons. The results demonstrate that the APG methodology can generate optimised robotic programs across all cases, confirming its effectiveness as a generalised framework for automated robot programming.

The reinforcement learning (RL) training behaviour provides important insights into the optimisation process. In all use cases, the training initially exhibits high variability in reward values, reflecting the exploratory nature of the agent as it evaluates different combinations of parameterised recipes. This phase is particularly pronounced in more complex tasks, where the number of possible recipe combinations is significantly larger. As training progresses, the agent begins to identify

Table 5

R3M environment launch results in simulation, reporting success rate and timing statistics for the main launch components and overall environment initialisation.

Element	Iterations	Success Rate	Mean (s)	Median (s)	STD (s)	Min/Max (s)
Robot Spawn	1700	100.00%	1.91	1.90	0.09	1.32
Robot Controllers	1700	100.00%	8.46	8.40	0.23	1.59
Robot Operation	1700	100.00%	8.46	8.40	0.23	1.59
Robot Monitoring	1700	100.00%	7.44	7.41	0.12	1.37
R3M Environment	6700	99.93%	8.54	8.50	0.20	1.14
APG ROS 2 Node	5000	100.00%	0.54	0.52	0.04	0.22

Table 6

R3M environment launch results for the ABB robot cell, reporting success rate and timing statistics for robot connection, ROS 2 driver initialisation, core launch components, and overall environment initialisation.

Element	Iterations	Success Rate	Mean (s)	Median (s)	STD (s)	Min/Max (s)
Robot Connection	500	99.80%	1.04	0.58	2.34	23.17
ROS 2 Driver	500	99.20%	1.95	0.61	3.56	23.08
Robot Controllers	500	99.20%	8.56	7.17	3.79	22.46
Robot Operation	500	99.20%	8.56	7.17	3.79	22.46
Robot Monitoring	500	99.20%	7.63	6.26	3.80	23.35
R3M Environment	1600	99.69%	7.71	7.20	1.62	16.63
APG ROS 2 Node	1100	100.00%	0.78	0.75	0.08	0.25

Table 7

R3M environment launch results for the UR robot cell, reporting success rate and timing statistics for robot connection, ROS 2 driver initialisation, core launch components, and overall environment initialisation.

Element	Iterations	Success Rate	Mean (s)	Median (s)	STD (s)	Min/Max (s)
Robot Connection	500	100.00%	2.78	2.81	0.22	1.38
ROS 2 Driver	500	100.00%	2.78	2.81	0.22	1.38
Robot Controllers	500	100.00%	9.22	9.22	0.23	1.43
Robot Operation	500	100.00%	9.22	9.22	0.23	1.43
Robot Monitoring	500	100.00%	8.32	8.33	0.23	1.44
R3M Environment	1600	99.88%	11.00	11.08	0.28	1.73
APG ROS 2 Node	1100	100.00%	0.78	0.75	0.08	0.25

Table 8

R3M Architecture test results: R3M Skill Execution (ABB Robot).

Scenario (ABB)	Iterations	Success Rate	Mean (s)	Median (s)	STD (s)	Min/Max (s)
Simulation	1000	100.00%	4.79	4.79	0.01	0.05
Real Robot	1000	100.00%	4.00	3.47	0.66	1.40

Table 9

R3M Architecture test results: R3M Skill Execution (UR Robot).

Scenario (UR)	Iterations	Success Rate	Mean (s)	Median (s)	STD (s)	Min/Max (s)
Simulation	1000	100.00%	13.14	13.14	0.01	0.05
Real Robot	1000	100.00%	13.28	13.29	0.01	0.06

Table 10

R3M Architecture test results: R3M Use-Case Execution (ABB Robot).

Scenario	Iterations	Success	Liaisons	Mean (s)	Median (s)	STD (s)	Min/Max (s)
Simulation	1000	97.40%	96.99%	72.07	72.07	0.05	0.33
Real Robot	100	100.00%	100.00%	58.70	58.70	0.04	0.20

Table 11

R3M Architecture test results: R3M Use-Case Execution (UR Robot).

Scenario	Iterations	Success	Liaisons	Mean (s)	Median (s)	STD (s)	Min/Max (s)
Simulation	1000	97.10%	96.70%	51.49	51.48	0.02	0.16
Real Robot	100	100.00%	100.00%	77.13	77.12	0.065	0.30

more efficient execution strategies, leading to a steady increase in reward values, which corresponds to a reduction in execution time. In the final stage of training, all use cases demonstrate clear convergence, where the reward stabilises at its maximum value. Notably, the final converged reward represents the best performance achieved during training, with no intermediate episode exceeding this value as shown in Fig. 13. This confirms that the learned policy corresponds to the optimal selection of recipes within the explored search space.

A key observation from the results is the variation in convergence behaviour across different use cases. Simpler tasks, such as LamSheet Pick and Place, converge relatively quickly due to the limited number of skill requirements and recipe alternatives. In contrast, more complex scenarios, such as Cylinder Stacking and Panel Handling, require a larger number of training episodes to achieve convergence. This is primarily due to the increased number of decision points and the presence of multiple feasible recipes for each skill requirement, which expands

the action space. Additionally, tasks involving multiple liaisons introduce more stringent success conditions, requiring the agent to learn coordinated sequences of actions rather than isolated skill executions. These differences are clearly reflected in the RL training curves, where more complex tasks exhibit longer exploration phases and slower convergence rates.

The results also highlight the importance of using task-level performance metrics for evaluation. The reward function, defined as the negative execution time, ensures that the optimisation process directly targets the efficiency of the generated robot program. As a result, convergence of the RL agent corresponds to minimisation of total execution time as shown in Fig. 13. Across all use cases, the final policies consistently achieve lower execution times by selecting more efficient parametrised recipes and avoiding unnecessary or suboptimal operations. The reduction in variability observed in the later stages of training, as indicated by the narrowing confidence bands, further confirms the stability and reliability of the learned policies.

It is important to note that the reward values remain negative throughout the training process, as they represent execution time penalties by design. Consequently, convergence is characterised by an increase in reward values (i.e., reduction in penalty magnitude) rather than a decrease. This interpretation ensures consistency between the mathematical formulation and the observed training behaviour. Overall, the results demonstrate that the APG methodology provides a robust and scalable solution for automated robot programming. By combining structured domain modelling with reinforcement learning-based optimisation, the approach can adapt to different assembly scenarios and consistently generate efficient and executable robotic programs, thereby validating its applicability across a wide range of industrial use cases.

6.3. R3M perception: one-shot detection and MegaPose-6D

The perception system was evaluated under 2 different conditions executed within R3M architecture: multi-object detection in simulation and single-object detection in a real robot cell. For multi-object detection in simulation, three objects were randomly selected within a

Gazebo-simulated environment. Their poses were estimated and recorded, with this process repeated 200 times per scene. For single-object detection in a real robot cell, the object was held by UR5e with a robotic gripper. In practical robotic manipulation scenarios, objects frequently become partially occluded by the gripper during grasping, necessitating an evaluation of perception performance under such conditions. The 6-DoF pose of the object in the world coordinate system was determined using the forward kinematics of robot, along with the camera's intrinsic and extrinsic parameters which are obtained by calibration. These computed poses served as the ground truth, and the experiment was repeated 100 times per object. As the evaluation was focused on the perception itself, the human verification was disabled, which means every result of One-Shot Detection was assumed valid to be the input of 6DoF estimation. To evaluate the accuracy of the estimated poses, we adopt two metrics: the Euclidean distance in 3D space for translation error, and the angular deviation in roll, pitch, and yaw based on Euler angles for rotation error. To properly handle rotationally symmetric objects, a symmetry-aware evaluation strategy is employed, following the formulations described below [44]:

$$S(u) = \{S \in SE(3) | \forall T \in SE(3), R(u, T) = R(u, TS)\}$$

where $S(u)$ is a set of poses of an object u that can be symmetrical poses for each other, T represents a 6D pose and R indicates the 2D projection for a given 6D pose, which means all the poses that results in an identical 2D appearance will be considered the equivalent poses.

To assess the perception system's performance in the Cube Kitting and Cylinder Stacking use cases, ten representative geometries were evaluated in both simulated and real-world environments. These geometries, commonly encountered in industrial assembly scenarios, were selected to match the scale and shape characteristics of the cube and cylinder used in the target use cases. For each geometry, 200[ZZ1.1] random 6-DoF poses were generated in three-dimensional space. The estimated poses produced by the perception system were compared against ground truth using translation error (measured in centimetres) and rotation error (measured in degrees) to quantitatively evaluate accuracy.

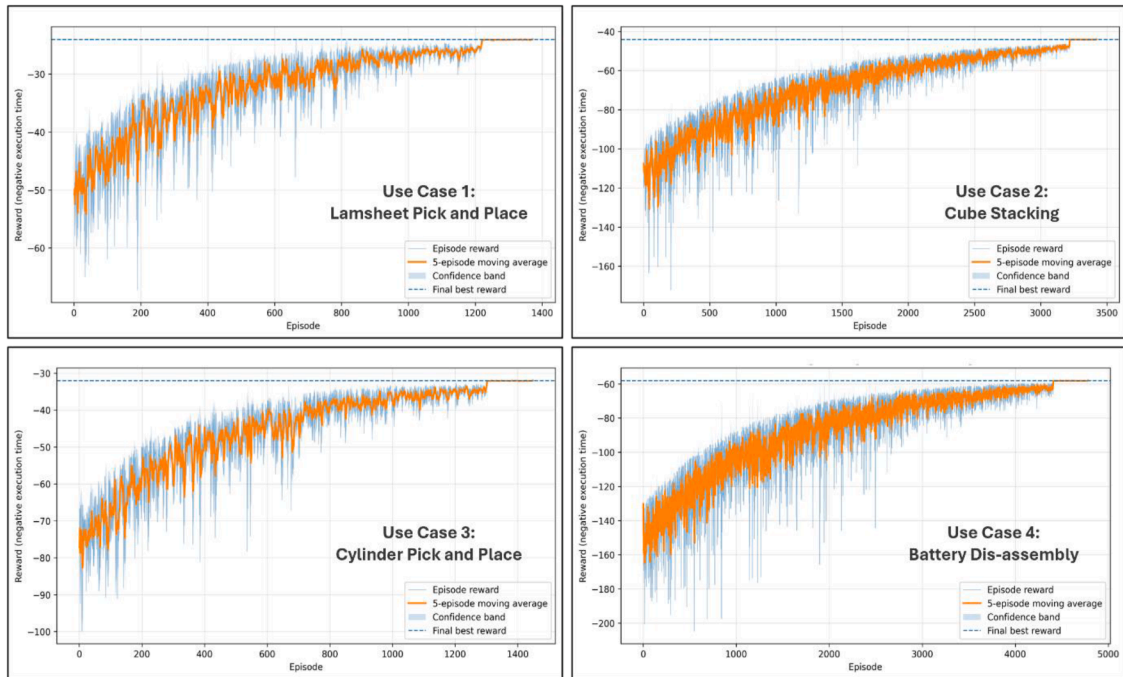


Fig. 13. Reinforcement learning training results across the evaluated use cases. The reward curves compare policy convergence behaviour under different task conditions, indicating how repeated interaction with the digital twin enables the agent to reduce execution time penalties through improved selection of parametrised recipes.

The detailed evaluation metrics for perception system are presented in Table 12:

The experimental results demonstrate that the perception system performs well under normal conditions, achieving low translation and rotation errors for objects with distinctive features and minimal ambiguity, such as adapter plate (triangular), connector (planar), and star. However, pose accuracy was sensitive to object scale and geometric ambiguity. In particular, the screw produced the largest orientation errors due to its small size, and the car rim showed a relatively high rejection rate and yaw error due to its rotational symmetry. Tests conducted in the real robotic cell maintained high confidence and reduced rejection rates for the selected components, although their rotational errors and variability generally increased compared with simulation. Overall, all objects exhibited small translational localisation errors, while rotation estimation was less accurate for symmetric objects and those with ambiguous geometric features.

As shown in Fig. 14, three representative failure modes of the perception system were identified. (a) Pseudo-symmetry occurs when viewpoint-dependent occlusion makes an asymmetric object appear symmetric. For the car rim, the spokes that distinguish its orientation may become invisible from certain viewpoints, while under a near-frontal view it is difficult to determine whether the spokes are located on the near or far side. A similar effect is observed for the star, whose directional serrations become less distinguishable as the object faces the

camera more directly. (b) Visually similar poses are illustrated by the triangular adapter plate. Although the object is not strictly flip-symmetric, poses separated by an approximately 180° rotation can produce nearly identical image appearances, making them difficult to distinguish from visual evidence alone. (c) High-uncertainty viewpoints occur for objects such as the planar connector and screw, where substantial changes in 3D orientation result in only minor changes in the projected image. This indicates that, in practice, manipulation strategies can be designed to remain valid across visually equivalent or similar poses, or pose hypotheses can be selected based only on the degrees of freedom relevant to the task. In addition, multi-view or active perception, together with uncertainty-triggered re-observation, can help reduce viewpoint-dependent ambiguity and improve the robustness of pose estimation.

6.4. Discussion

The evaluation indicates that the R3M framework can translate structured manufacturing knowledge into executable robotic behaviour while maintaining reliable operation across different robotic platforms and deployment conditions. In the Cube Kitting and Cylinder Stacking use cases, the UML and AML based product, process, and equipment models provided the basis for generating robot programmes through the APG pipeline, reducing the need for repeated manual coding. At the

Table 12

Quantitative Evaluation of Pose Estimation Accuracy for Various Objects. The orange coloured names are real-world trials.

OBJECT (name)		Confidence score	Reject@ 0.5 (%)	Delta 3D Distance (cm)	Roll (°)	Pitch (°)	Yaw (°)
cube		0.947±0.107	1.3	0.53 ± 0.42	1.78±3.04	2.19±3.92	1.93±4.79
cylinder		0.926±0.145	4.4	0.45 ± 0.31	2.10 ± 5.92	2.04 ± 3.94	0.56 ± 3.24
adapter plate triangular		0.995±0.034	0.0	0.32 ± 0.63	6.42±17.16	1.78 ± 5.49	6.03 ±26.03
planar connector		1.000±0.004	0.0	1.15 ± 4.33	8.44 ±32.49	1.68 ± 4.70	7.25 ±21.16
engine part cooler (round)		0.977±0.106	2.0	0.45 ± 0.98	8.00 ±25.69	4.76 ± 12.26	5.10 ±13.54
car rim		0.849±0.270	12.3	0.35 ± 0.65	6.42 ±17.16	5.96 ± 14.31	19.20±14.78
screw		0.943±0.209	1.8	0.54 ± 0.63	30.57 ±50.93	11.23 ±16.31	20.12±33.45
star		0.986±0.105	1.8	1.3 ± 0.22	1.83 ± 8.03	1.32 ± 5.46	0.66±5.45
engine part bearing		0.999±0.006	0.0	1.04 ± 1.68	9.96 ±32.29	3.39 ± 6.08	9.35±34.14
engine part cooler (round)		0.993±0.040	0.0	1.08 ± 1.48	17.70 ±37.96	6.78 ±11.59	7.30±17.32
car rim		0.913±0.180	5.5	0.76 ± 0.69	14.00 ±23.15	10.04 ±15.48	21.36±19.71

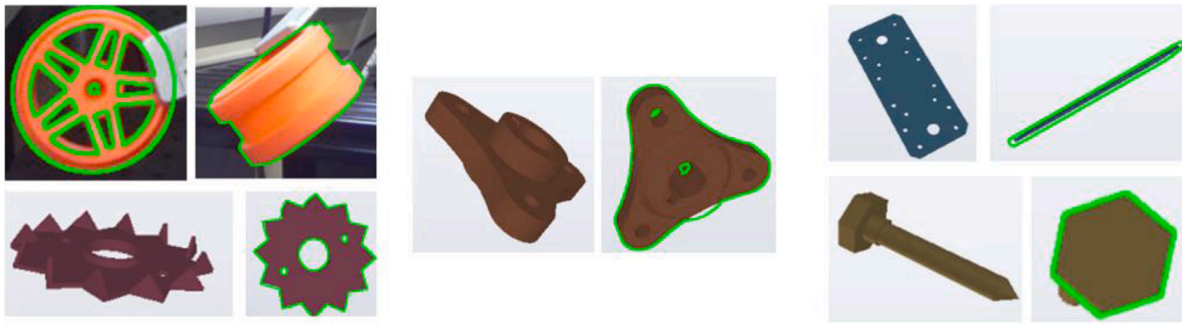


Fig. 14. Failure Modes of the Perception System: (a) Pseudo-Symmetry, (b) Visually Similar Poses, and (c) High-Uncertainty Viewpoints.

same time, the integrated perception modules supported object detection and six degree of freedom pose estimation under variable operating conditions, enabling task execution to remain linked to the observed state of the workcell. The consistently high success rates obtained during environment launch, skill execution, and complete programme execution in both simulation and real hardware demonstrate the robustness of the ROS 2 based architecture and its suitability for reconfigurable robotic manufacturing.

The principal contribution of the architecture lies in its model based interoperability. Rather than integrating isolated components through application specific wiring, the framework separates domain knowledge, skill selection, perception, and robot execution into modular but coordinated layers. This enabled skills, perception pipelines, robots, and cell configurations to be substituted without restructuring the core architecture. The transition between Cube Kitting and Cylinder Stacking, together with deployment in the RECON Cell, demonstrates that the same execution principles can be retained across different robots, hardware configurations, safety interfaces, and digital environments.

The evaluation also identifies limitations in APG scalability. Recipe generation becomes increasingly demanding as the number of skill alternatives, parameter choices, and feasible robot configurations increases. For a constrained six degree of freedom manipulator, a fully specified Cartesian pose is associated with a finite set of inverse kinematic solutions, subject to joint limits, singularities, and collision constraints. Where the task specifies only a position, partial orientation, or an admissible approach region, additional task level redundancy remains and must be resolved during planning. The current implementation mitigates this complexity through human defined candidate poses, but future work should incorporate data driven parameterisation, constraint aware inverse kinematic selection, and reuse of successful configurations learned from prior executions.

The reinforcement learning stage presents a further scalability challenge. As task length and the number of recipe alternatives increase, the action space expands and training may become less stable. Future APG implementations should therefore consider restructuring the learning problem rather than training a single flat policy over all available recipes. Hierarchical reinforcement learning can represent repeated manufacturing behaviours, such as grasp, move, align, place, and verify, as reusable subtasks or temporally extended actions [56]. In parallel, curriculum learning can train the agent progressively, beginning with simpler assemblies and gradually increasing task complexity as policy performance improves [57,58]. These strategies are particularly relevant to APG because many manufacturing operations share recurring skill patterns across different product variants and cell layouts.

The perception results confirm the value of combining one shot detection with CAD informed pose estimation for previously unseen objects. However, ambiguous object symmetry, partial occlusion, and reduced image fidelity remain limiting factors, particularly where single view observations cannot distinguish between visually equivalent poses. Future deployment should therefore consider multi view sensing, uncertainty estimation, and tighter coupling between perception

confidence and execution level decision making. In addition, real time performance remains sensitive to computational load, especially in scenarios requiring frequent perception updates, replanning, or coordination across multiple devices.

Overall, the R3M architecture provides a flexible and scalable model based framework for interoperable robotic manufacturing. It demonstrates that APG, perception, reinforcement learning, and ROS 2 execution can be combined within a coherent architecture that supports repeatable deployment, component substitution, and simulation to real hardware validation. Further work should focus on improving computational efficiency, formalising intermediate code representations for heterogeneous robot controllers, embedding execution monitoring into generated programmes, and extending learning and perception methods for larger and more uncertain industrial environments.

Table 13

Summary of the R3M Project open-source ROS 2 repositories and their main functionalities.

ros2_SimRealRobotControl	https://github.com/IFRA-Cranfield/ros2_SimRealRobotControl
Developed as part of the R3M Project to address the challenges of reconfigurability and generalisation in robotic-arm programming and control. It provides a standardised framework for modularity, interoperability and seamless integration of robots across different manufacturing setups. R3M-Cell builds on this architecture, enabling robotic cells at Cranfield University and AMRC-Sheffield to be simulated, configured and adapted for various tasks. The repository supports both simulation (Gazebo) and real-world deployment, with modules for multiple robots (e.g., ABB IRB-120, UR3) that handle simulation, control and motion-planning (MoveIt!2).	
R3M_Cell	https://github.com/IFRA-Cranfield/R3M_Cell
Contains the Gazebo and MoveIt!2 packages for the robotic cells deployed at Cranfield University and AMRC-Sheffield as part of the R3M initiative. Includes robot specifications and CAD models, URDF files for ROS 2, Gazebo simulation environments, MoveIt!2 integration and hardware-configuration information to bridge simulation and real-world implementation.	
R3M_APG	https://github.com/IFRA-Cranfield/R3M_APG
Contains all components of the Automatic Program Generation (APG) module of the R3M Project. It includes the main orchestrator that manages skill and program execution, reinforcement learning agent training, and R3M environment management. The repository also provides all ROS 2-based communication mechanisms that enable seamless integration of the APG modules within the R3M architecture.	
R3M_Perception	https://github.com/IFRA-Cranfield/R3M_Perception
Hosts the One-Shot Detection and MegaPose6D modules for object detection and pose estimation within the R3M architecture. Includes ROS 2 Nodes (Action Servers, Service & Topic handling) to manage execution of object detection (One-Shot) and pose estimation (M6D) workflows.	
R3M_ObjectPoseEstimation	https://github.com/IFRA-Cranfield/R3M_ObjectPoseEstimation
Contains YOLO + OpenCV-based object detection and pose estimation modules tailored for the R3M use-cases. Includes trained YOLOv11 models, ARUCO-based object pose estimation, and ROS 2 communication mechanisms that integrate the perception modules into the R3M architecture.	

7. Conclusions

This paper introduces, for the first time, a UML-based, ROS 2-driven integration framework that unifies domain-level modelling with execution-level control to enable APG, modular perception, and reinforcement learning for reconfigurable robotic manufacturing. By coupling formal UML representations of product, process, and equipment with ROS 2 communication and control mechanisms, the framework bridges abstract system knowledge and concrete robot behaviour. In doing so, it addresses persistent fragmentation in the literature, where prior efforts typically concentrate on middleware, perception, or hardware modularity in isolation, and advances the goals of Industry 5.0 through intelligent, collaborative, and sustainable production.

The framework was empirically validated through two representative industrial use cases, Cube Kitting and Cylinder Stacking, deployed on distinct robotic platforms at Cranfield University and the AMRC. The results demonstrate scalability across sites and platforms, a marked reduction in manual programming effort through APG, and reliable perception via a plug-in pipeline. The modular design enabled component substitution without architectural refactoring, evidencing practical adaptability to heterogeneous environments. At the same time, observations on computational load and real-time responsiveness indicate avenues for optimisation to further strengthen robustness under stringent operational constraints.

Overall, the research establishes a principled foundation for robotic systems that are both technically robust and semantically interoperable. Integrating UML-based domain models with ROS 2 execution provides a novel pathway to align high-level intent with dependable low-level control. Future work will focus on improving computational efficiency, extending APG and perception capabilities, formalising intermediate representations for multi-robot and multi-vendor deployments, and conducting large-scale industrial trials. Pursuing these directions will consolidate the framework as a cornerstone for adaptive, scalable, and autonomous robotic manufacturing aligned with the long-term vision of reconfigurable production and Industry 5.0.

Source code

All components of the R3M Project's ROS 2 software stack will be

Appendix A. Detailed R3M Perception System Workflows

This appendix provides detailed representations of the interactions within the R3M perception system. Fig. Ap.1 presents the activity-level workflow, from the initial architecture trigger and object detection to HMI-based bounding-box validation, operator correction, and 6DoF pose estimation. Fig. Ap.2 expands this representation at sequence level, showing the exchanges between the operator, HMI, R3M architecture, camera, One-Shot Detection module, MegaPose coarse estimator, and MegaPose refiner.

released open-source once this paper is published, and will be made available via the IFRA-Cranfield's GitHub page (<https://github.com/IFRA-Cranfield>, see Table 13). These packages were developed and tested on Ubuntu 22.04 with ROS 2 Humble Hawksbill. The repositories include full documentation, installation instructions and all resources needed to replicate the ROS 2 environments and features described in this paper.

CRediT authorship contribution statement

Seemal Asif: Writing – review & editing, Writing – original draft, Supervision, Project administration, Methodology, Investigation, Funding acquisition, Conceptualization. **Mikel Bueno:** Writing – review & editing, Writing – original draft, Visualization, Software, Methodology, Formal analysis, Data curation, Conceptualization. **Paul Anandan:** Writing – original draft, Validation, Software, Methodology, Data curation. **Pedro Ferreira:** Writing – review & editing, Resources, Investigation. **Ze Zhang:** Writing – original draft, Visualization, Validation, Software, Methodology, Data curation. **Yue Yao:** Writing – original draft, Visualization, Validation, Software, Methodology. **Windo Hutabarat:** Supervision, Resources. **Gautham Ragunathan:** Writing – original draft, Validation, Methodology, Data curation. **Ashutosh Tiwari:** Supervision, Investigation. **Niels Lohse:** Writing – review & editing, Supervision, Resources, Investigation. **Phil Webb:** Resources, Project administration, Investigation. **Lloyd L.C. Tinkler:** Resources, Investigation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This research is supported by EPSRC grant (EP/V051180/1) for the Reconfigurable Robotics for Responsive Manufacture - R3M.

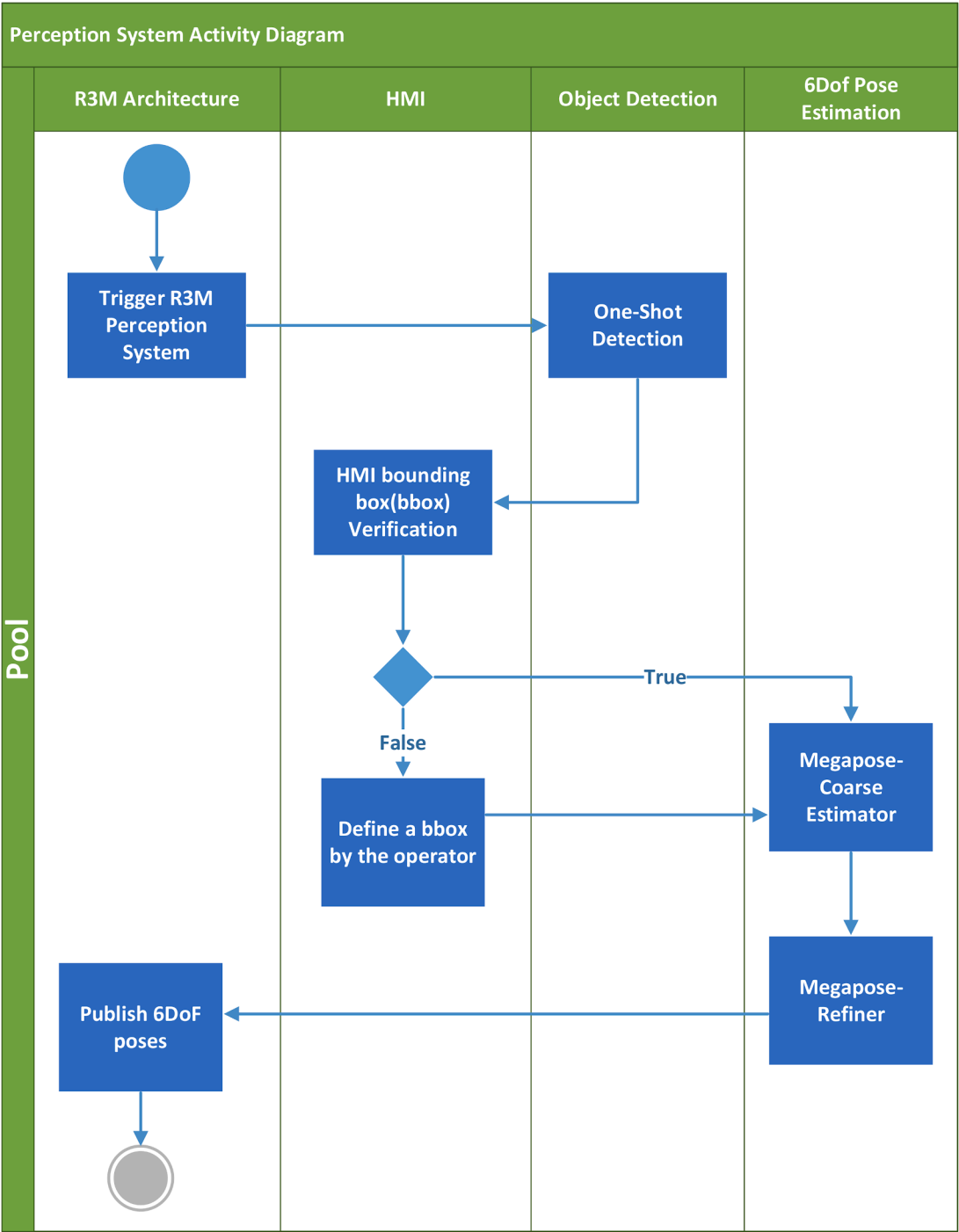


Fig. A.1. Activity flow of the R3M perception system, covering the transition from architecture-level triggering to object detection, HMI-based bounding-box validation, operator correction where required, and subsequent six-degree-of-freedom pose estimation. The workflow integrates one-shot detection with MegaPose coarse estimation and refinement to support reliable localisation of previously unseen objects.

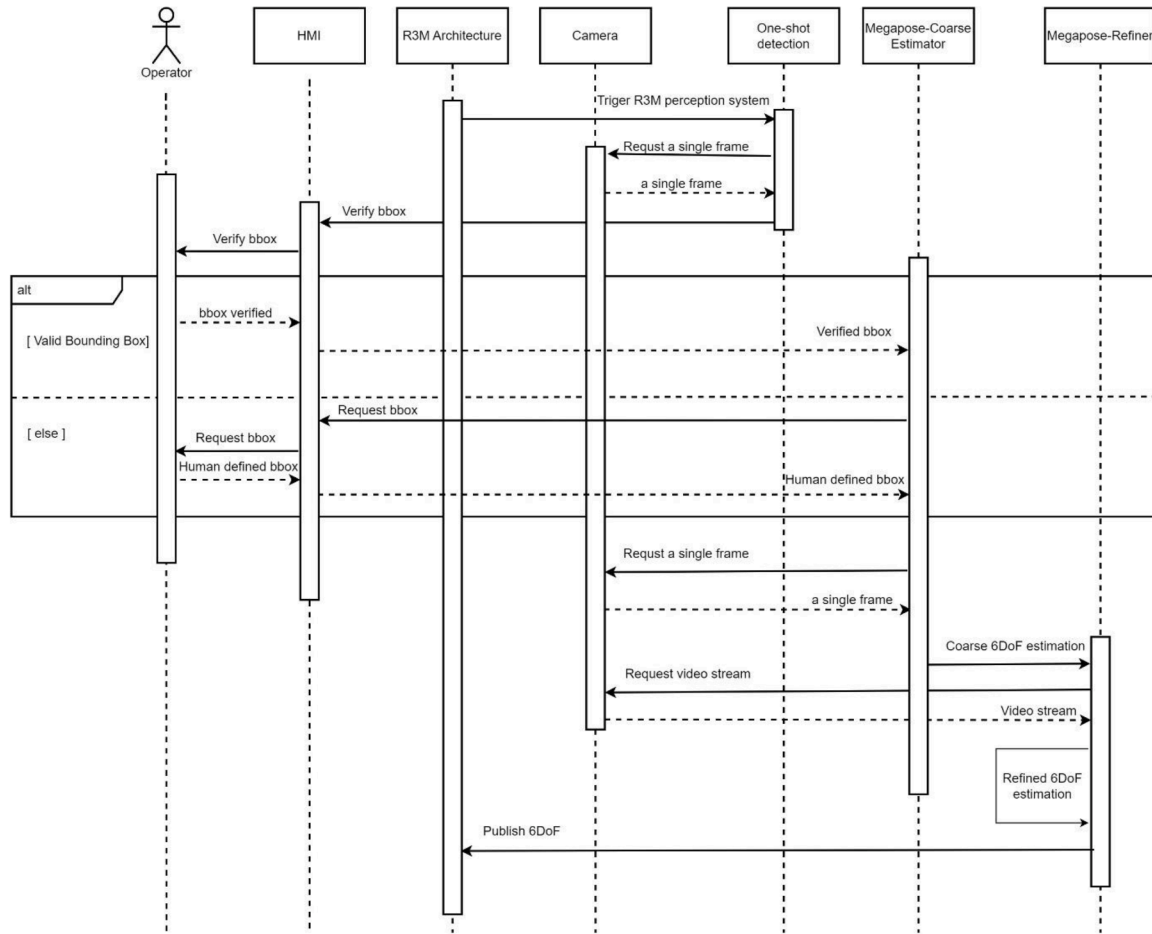


Fig. A.2. Sequence-level representation of perception-system interactions among the operator, HMI, R3M architecture, camera, One-Shot Detection module, MegaPose coarse estimator, and MegaPose refiner. The sequence captures the exchange of image-acquisition requests, bounding-box validation, manual correction, pose estimation, iterative refinement, and final pose publication for downstream robotic execution.

Appendix B. R3M Configuration and Message Structures

This appendix provides implementation-level details of the structured configuration files and ROS 2 messages used by the R3M execution environment. Fig. Ap.3 presents examples of the .yaml files used to configure the robot cell, use-case information, and robot skill recipes. Fig. Ap.4 presents the custom ROS 2 message structures used to communicate robot, product, pose, and liaison information during skill execution and feedback.

Robot Cell Information	Use-Case Information	Robot Skill: Recipe
cell.yaml <pre> ID: "r3mcell_01" Name: "Cell 01 - Cranfield" urdf: "r3mcell_01.urdf" Robot: "ABB IRB-120" EE: "Schunk EGP-64" </pre>	r3m_01.yaml <pre> information: use-case: "CubeStacking" name: "r3m_01" Robot: model: "ABB IRB-120" EE: "Schunk EGP-64" InitPose: "{}" Objects: [Name, CAD, InitPose] Liaisons: [Name, Parent/Child, Cond.] </pre>	12.yaml <pre> information: use-case: "r3m_01" recipe: "To HomePos" id: 12 type: "PTP" speed: 0.1 pose: position: (x,y,z) orient.: (qx,qy,qz,qw) </pre>

Fig. A.3. Example .yaml configuration files used to initialise the R3M environment. The Robot Cell Information file defines the robot and end-effector configuration, including references to the corresponding description and control files. The Use-Case Information file defines the product/process context, including cell layout, object information, and recipe references. The Robot Skill Recipe file specifies the executable skill logic and associated parameters used during program execution. Together, these files provide the structured information required by the orchestrator to configure, launch, and operate the system.

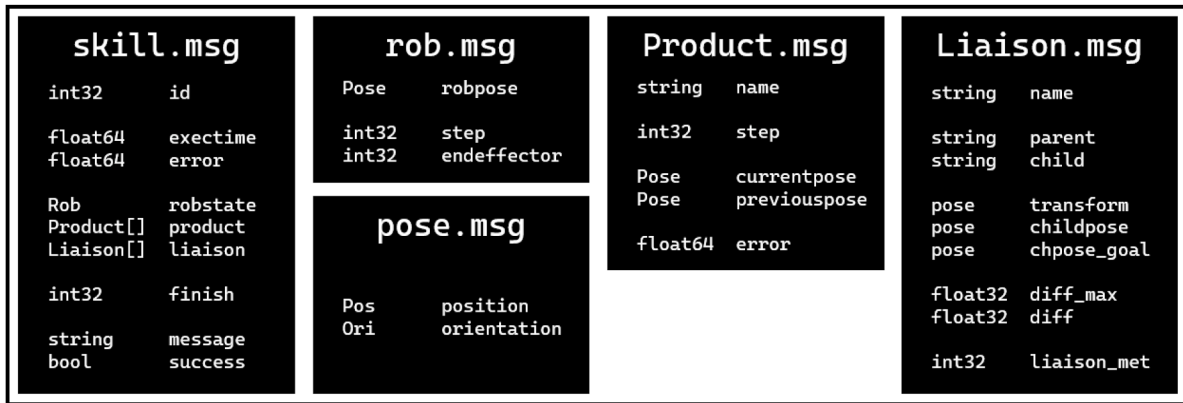


Fig. A.4. ROS 2 message structures used for R3M robot skill execution and feedback. The figure shows the custom message types that represent robot state, product state, pose information, and liaison conditions during execution. These structures provide the common data model used by the orchestrator during recipe execution and are aligned with the robot cell, use-case, and skill information represented in the .yaml files introduced in Section 4.5.1. System Components.

Data availability

Data will be made available on request.

References

- [1] T. Gašpar, et al., Smart hardware integration with advanced robot programming technologies for efficient reconfiguration of robot workcells, *Robot. Comput. Integr. Manuf.* 66 (2020) 101979, <https://doi.org/10.1016/j.rcim.2020.101979>.
- [2] B. Wen, W. Yang, J. Kautz, and S. Birchfield, "FoundationPose: Unified 6D Pose Estimation and Tracking of Novel Objects," 2024, [Online]. Available: <http://arxiv.org/abs/2312.08344>.
- [3] M.G. Mehrabi, A.G. Ulsoy, Y. Koren, Reconfigurable manufacturing systems: key to future manufacturing, *J. Intell. Manuf.* 11 (4) (2000) 403–419, <https://doi.org/10.1023/A:1008930403506>.
- [4] J. Morgan, M. Halton, Y. Qiao, J.G. Breslin, Industry 4.0 smart reconfigurable manufacturing machines, *J. Manuf. Syst.* 59 (2021) 481–506, <https://doi.org/10.1016/j.jmsys.2021.03.001>.
- [5] R. Arista, F. Mas, D. Morales-Palma, C. Vallellano, An ontology-based engineering methodology applied to aerospace reconfigurable manufacturing systems design, *Int. J. Prod. Res.* 62 (6) (2024) 2286–2304, <https://doi.org/10.1080/00207543.2023.2217296>.
- [6] D. Bozhinoski, E. Aguado, M.G. Oviedo, C. Hernandez, R. Sanz, A. Wasowski, A modeling tool for reconfigurable skills in ROS, in: *Proceedings of the 2021 IEEE/ACM 3rd International Workshop on Robotics Software Engineering, RoSE 2021*, 2021, pp. 25–28, <https://doi.org/10.1109/ROSE52553.2021.00011>.
- [7] F. Mo, et al., A framework for manufacturing system reconfiguration and optimisation utilising digital twins and modular artificial intelligence, *Robot. Comput. Integr. Manuf.* 82 (2023) 102524, <https://doi.org/10.1016/j.rcim.2022.102524>.
- [8] S. Pandurevic, Z. Wang, D. Sanderson, S. Ratchev, Digital Twin enabled layout optimization framework for photogrammetry cameras in reconfigurable robotic work cells, in: *Advanced Manufacturing, 2*, American Society of Mechanical Engineers, 2024, <https://doi.org/10.1115/IMECE2024-145273>.
- [9] L. Wang, Z. Wang, K. Gumma, A. Turner, S. Ratchev, Multi-agent cooperative swarm learning for dynamic layout optimisation of reconfigurable robotic assembly cells based on digital twin, *J. Intell. Manuf.* 36 (2024) 2959–2982, <https://doi.org/10.1007/s10845-023-02229-7>.
- [10] M. Deniša, A. Ude, Improving ergonomics of collaborative robot workcells using passive reconfigurable fixtures, *IEEE Access* 12 (2024) 124871–124888, <https://doi.org/10.1109/ACCESS.2024.3451975>.
- [11] M. Simonić, et al., An application study on reconfigurable robotic workcells and policy adaptation for electronic waste recycling, in: *Proceedings of the 2024 IEEE International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE International Conference on Robotics, Automation and Mechatronics (RAM)*, IEEE, 2024, pp. 180–186, <https://doi.org/10.1109/CIS-RAM61939.2024.10673362>.
- [12] S. Wang, J. Wan, D. Zhang, D. Li, C. Zhang, Towards smart factory for industry 4.0: a self-organized multi-agent system with big data based feedback and coordination, *Comput. Netw.* 101 (2016) 158–168, <https://doi.org/10.1016/j.comnet.2015.12.017>.
- [13] D. Guri, M. Lee, O. Kroemer, and G. Kantor, "Hefty: a modular reconfigurable robot for advancing robot manipulation in agriculture," 2024, Accessed: Apr. 08, 2025. [Online]. Available: <https://arxiv.org/abs/2402.18710v1>.
- [14] Y. Dai, S. He, X. Nie, X. Rui, S. Li, S. He, Research on reconfiguration strategies for self-reconfiguring modular robots: a review, *J. Intell. Robot. Syst.* 110 (2) (2024) 47, <https://doi.org/10.1007/s10846-024-02067-6>.
- [15] A.R. Yelles-Chaouche, E. Gurevsky, N. Brahimi, A. Dolgui, Reconfigurable manufacturing systems from an optimisation perspective: a focused review of literature, *Int. J. Prod. Res.* 59 (21) (2021) 6400–6418, <https://doi.org/10.1080/00207543.2020.1813913>.
- [16] V. Ashiwal, A. Zoitl, M. Konnerth, A service bus concept for modular and adaptable PLC-software, in: *Proceedings of the IEEE Symposium on Emerging Technologies and Factory Automation, ETFA 2020*, 2020, pp. 22–29, <https://doi.org/10.1109/ETFA46521.2020.9211908>. -September.
- [17] E. Trunzer, et al., System architectures for Industrie 4.0 applications, *Prod. Eng.* 13 (3–4) (2019) 247–257, <https://doi.org/10.1007/s11740-019-00902-6>.
- [18] P. Pahlevannejad, A. Herget, R. Moreno, A. Hennecke, M. Ruskowski, Implementation and testing of a modular system architecture for generic hybrid production cells in an industrial environment, in: *Proceedings of the 45th Annual Conference of the IEEE Industrial Electronics Society, IECON 2019*, IEEE, 2019, pp. 4229–4234, <https://doi.org/10.1109/IECON.2019.8926861>.
- [19] S.C. Jepsen, B. Siewertsen, T. Worm, A reconfigurable industry 4.0 middleware software architecture, in: *Proceedings of the 2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*, IEEE, 2023, pp. 1–11, <https://doi.org/10.1109/ICSA-C57050.2023.00023>.
- [20] M. Quigley et al., "ROS: an open-source robot operating system," 2009.
- [21] A. Bonci, F. Gaudeni, M.C. Giannini, S. Longhi, Robot Operating System 2 (ROS2)-based frameworks for increasing robot autonomy: a survey, *Appl. Sci.* 13 (23) (2023) 12796, <https://doi.org/10.3390/app132312796>.
- [22] M. Simonic, et al., Modular ROS-based software architecture for reconfigurable, Industry 4.0 compatible robotic workcells, in: *Proceedings of the 2021 20th International Conference on Advanced Robotics (ICAR)*, IEEE, 2021, pp. 44–51, <https://doi.org/10.1109/ICAR53236.2021.9659378>.
- [23] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, W. Woodall, Robot Operating System 2: design, architecture, and uses in the wild, *Sci. Robot.* 7 (66) (2022), <https://doi.org/10.1126/scirobotics.abm6074>.
- [24] J. Martin, et al., A generic ROS-based control architecture for pest inspection and treatment in greenhouses using a mobile manipulator, *IEEE Access* 9 (2021) 94981–94995, <https://doi.org/10.1109/ACCESS.2021.3093978>.
- [25] A. Testa, A. Camisa, G. Notarstefano, ChoiRbot: a ROS 2 toolbox for cooperative robotics, *IEEE Robot. Autom. Lett.* 6 (2) (2021) 2714–2720, <https://doi.org/10.1109/LRA.2021.3061366>.
- [26] M. Ferrari, et al., A distributed control architecture for a multi-agent robotic cell: a battery pack disassembly case study, *Int. J. Comput. Integr. Manuf.* 39 (2025) 1–21, <https://doi.org/10.1080/0951192X.2025.2531488>.
- [27] P. Leitão, J. Barbosa, A. Pereira, J. Barata, A.W. Colombo, Specification of the PERFORM architecture for the seamless production system reconfiguration, in: *Proceedings of the IECON (Industrial Electronics Conference)*, 2016, pp. 5729–5734, <https://doi.org/10.1109/IECON.2016.7793007>.
- [28] H. Medellin, J. Corney, J. Ritchie, T. Lim, Automatic generation of robot and manual assembly plans using octrees, *Assem. Autom.* 30 (2) (2010) 173–183, <https://doi.org/10.1108/01445151011029817>.
- [29] R. S.-P. of the S., and undefinedcostar. hw. ac. uk J. Ritchie, T. Lim, Generation of assembly process plans and associated Gbireth motion study data, in: J.M. Ritchie, T. Lim, R.S. Sung, H. Medell (Eds.), *Proceedings of the Second Virtual Manufacturing Workshop, EU Intuition*, 2008, 2008•costar. hw. ac. uk, Accessed: Jun. 12, 2025. [Online]. Available: https://www.costar.hw.ac.uk/publications/Ritchie_Lim_Sung_Medellin_VIRMAN08.pdf.
- [30] M. Babciński, F. Cruz, N. Duarte, S. Santos, S. Alves, P. Neto, Offline robot programming assisted by task demonstration: an AutomationML interoperable solution for glass adhesive application and welding, *Int. J. Comput. Integr. Manuf.* 38 (2024) 864–875, <https://doi.org/10.1080/0951192X.2024.2358042>.
- [31] L.A. Ferreira, Y.L. Figueira, I.F. Iglesias, M.A. Souto, Offline CAD-based robot programming and welding parametrization of a flexible and adaptive robotic cell

- using enriched CAD/CAM system for shipbuilding, *Procedia Manuf.* 11 (2017) 215–223, <https://doi.org/10.1016/j.promfg.2017.07.228>.
- [32] P. Danny, P. Ferreira, N. Lohse, M. Guedes, An AutomationML model for plug-and-produce assembly systems, in: *Proceedings of the 2017 IEEE 15th International Conference on Industrial Informatics, INDIN 2017*, 2017, pp. 849–854, <https://doi.org/10.1109/INDIN.2017.8104883>.
- [33] T. Lohi, S. Soutukorva, T. Heikkilä, Programming of skill-based robots, in: *Proceedings of the 2024 IEEE 19th Conference on Industrial Electronics and Applications, ICIEA 2024*, 2024, <https://doi.org/10.1109/ICIEA61579.2024.10664981>.
- [34] M. Zimmer, A. Al-Yacoub, P. Ferreira, N. Lohse, Trajectory creation towards fast skill deployment in plug-and-produce assembly systems: a gaussian-mixture model approach, in: *Proceedings of the UK-RAS19 Conference: "Embedded Intelligence: Enabling and Supporting RAS Technologies"*, 2019, <https://doi.org/10.3125/ukras19.23>.
- [35] T. Inoue, G. De Magistris, A. Munawar, T. Yokoya, R. Tachibana, Deep reinforcement learning for high precision assembly tasks, in: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, 2017, <https://doi.org/10.1109/IROS.2017.8202244>.
- [36] S. Asif, et al., Rapid and automated configuration of robot manufacturing cells, *Robot. Comput. Integr. Manuf.* 92 (2025) 102862. Accessed: Jun. 12, 2025. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0736584524001492>.
- [37] A. MacAluso, N. Cote, S. Chitta, Toward automated programming for robotic assembly using ChatGPT, in: *Proceedings of the IEEE International Conference on Robotics and Automation*, 2024, pp. 17687–17693, <https://doi.org/10.1109/ICRA57147.2024.10610554>.
- [38] H. You, Y. Ye, T. Zhou, Q. Zhu, J. Du, Robot-enabled construction assembly with automated sequence planning based on ChatGPT: RoboGPT, *Buildings* 13 (7) (2023) 1772, <https://doi.org/10.3390/buildings13071772>.
- [39] S.M. Marvasti-Zadeh, L. Cheng, H. Ghanei-Yakhdan, S. Kasaei, Deep learning for visual tracking: a comprehensive survey, *IEEE Trans. Intell. Transp. Syst.* 23 (5) (2022) 3943–3968, <https://doi.org/10.1109/TITS.2020.3046478>.
- [40] R. Kaur, S. Singh, A comprehensive review of object detection with deep learning, *Digit. Signal Process.* 132 (2023) 103812, <https://doi.org/10.1016/j.dsp.2022.103812>.
- [41] G. Marullo, L. Tanzi, P. Piazzolla, E. Vezzetti, 6D object position estimation from 2D images: a literature review, *Multimed. Tools Appl.* 82 (16) (2023) 24605–24643, <https://doi.org/10.1007/s11042-022-14213-z>.
- [42] L. Lipson, Z. Teed, A. Goyal, and J. Deng, "Coupled Iterative Refinement for 6D Multi-Object Pose Estimation," 2022.
- [43] R.L. Haugaard and A.G. Buch, "SurfEmb: Dense and Continuous Correspondence Distributions for Object Pose Estimation with Learnt Surface Embeddings," 2022, [Online]. Available: <http://arxiv.org/abs/2111.13489>.
- [44] Y. Labbé, J. Carpentier, M. Aubry, and J. Sivic, "CosyPose: consistent multi-view multi-object 6D pose estimation," 2020.
- [45] D. Horváth, G. Erdős, Z. Isteneš, T. Horváth, S. Földi, Object detection using Sim2Real domain randomization for robotic applications, *IEEE Trans. Robot.* 39 (2) (2023) 1225–1243, <https://doi.org/10.1109/TRO.2022.3207619>.
- [46] M. Sundermeyer et al., "Multi-path Learning for Object Pose Estimation Across Domains," 2020, [Online]. Available: <http://arxiv.org/abs/1908.00151>.
- [47] I. Shugurov, F. Li, B. Busam, S. Ilic. OSOP: A Multi-Stage One Shot Object Pose Estimation Framework, *IEEE*, 2022, pp. 6825–6834, <https://doi.org/10.1109/CVPR52688.2022.00671>.
- [48] V.N. Nguyen, Y. Hu, Y. Xiao, M. Salzmann, and V. Lepetit, Templates for 3D Object Pose Estimation Revisited: Generalization to New Objects and Robustness to Occlusions," 2022, [Online]. Available: <http://arxiv.org/abs/2203.17234>.
- [49] Y. Labbé et al., "MegaPose: 6D pose estimation of novel objects via render & compare," 2022.
- [50] M.G. Mehrabi, A.G. Ulsoy, Y. Koren, Reconfigurable manufacturing systems and their enabling technologies, *Int. J. Manuf. Technol. Manag.* 1 (1) (2000) 114, <https://doi.org/10.1504/IJMTM.2000.001330>.
- [51] A. MacAluso, N. Cote, S. Chitta, Toward automated programming for robotic assembly using ChatGPT, in: *Proceedings of the IEEE International Conference on Robotics & Automation*, 2024, pp. 17687–17693, <https://doi.org/10.1109/ICRA57147.2024.10610554>.
- [52] S. Chitta, Moveit!: an introduction, *Stud. Comput. Intell.* 625 (2016) 3–27, https://doi.org/10.1007/978-3-319-26054-9_1/FIGURES/14.
- [53] M. Bueno, I. Bernardino, S. Asif, P. Webb, Unlocking the potential of robot manipulators: seamless integration framework, in: *Proceedings of the IEEE International Conference on Automation Science and Engineering*, 2024, pp. 2610–2617, <https://doi.org/10.1109/CASE59546.2024.10711476>.
- [54] M.B. Viso, J. Huang, S. Asif, F. Khan, P. Webb, Towards robot software abstraction: ROS 2-based framework for object handling within a robot cell, in: *Proceedings of the 2024 IEEE 22nd International Conference on Industrial Informatics (INDIN)*, 2024, pp. 1–8, <https://doi.org/10.1109/INDIN58382.2024.10774307>.
- [55] M. Minderer et al., "Simple open-vocabulary object detection with vision transformers," 2022.
- [56] R.S. Sutton, D. Precup, S. Singh, Between MDPs and semi-MDPs: a framework for temporal abstraction in reinforcement learning, *Artif. Intell.* 112 (1–2) (1999) 181–211, [https://doi.org/10.1016/S0004-3702\(99\)00052-1](https://doi.org/10.1016/S0004-3702(99)00052-1).
- [57] Y. Bengio, J. Louradour, R. Collobert, J. Weston, Curriculum learning, in: *Proceedings of the 26th Annual International Conference on Machine Learning*, New York, NY, USA, ACM, 2009, pp. 41–48, <https://doi.org/10.1145/1553374.1553380>.
- [58] S. Narvekar, B. Peng, M. Leonetti, J. Sinapov, M.E. Taylor, and P. Stone, "Curriculum learning for reinforcement learning domains: a framework and survey," 2020.