Volume 32 | Number 1 | December 2026 ISSN 1387-3954

An FMI-centric open-source and interoperable framework for AI-driven robotic systems: a model-based methodology for integrating physical and AI models

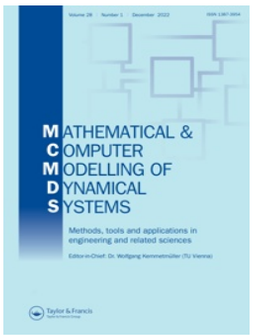
MATHEMATICAL & COMPUTER MODELLING OF DYNAMICAL SYSTEMS

Methods, tools and applications in engineering and related sciences

Editors-in-Chief: Wolfgang Kemmetmüller (TU Vienna)
Serkan Araci (HKU Türkiye)

Taylor & Francis
Taylor & Francis Group

MATHEMATICAL & COMPUTER MODELLING OF DYNAMICAL SYSTEMS
VOLUME 32 NUMBER 1 DECEMBER 2026



Taylor & Francis
Taylor & Francis Group

Mathematical and Computer Modelling of Dynamical Systems

Methods, Tools and Applications in Engineering and Related Sciences

ISSN: 1387-3954 (Print) 1744-5051 (Online) Journal homepage: www.tandfonline.com/journals/nmcm20

An FMI-centric open-source and interoperable framework for ai-driven robotic systems: a model-based methodology for integrating physical and AI models

Sebastian Rojas-Ordoñez , Mikel Segura , Jorge Arboleda & Ekaitz Zulueta

To cite this article: Sebastian Rojas-Ordoñez , Mikel Segura , Jorge Arboleda & Ekaitz Zulueta (2026) An FMI-centric open-source and interoperable framework for ai-driven robotic systems: a model-based methodology for integrating physical and AI models, Mathematical and Computer Modelling of Dynamical Systems, 32:1, 2707718, DOI: 10.1080/13873954.2026.2707718

To link to this article: https://doi.org/10.1080/13873954.2026.2707718

© 2026 The Author(s). Published by Informa UK Limited, trading as Taylor & Francis Group.

Published online: 30 Jul 2026.

Submit your article to this journal

Article views: 346

View related articles

View Crossmark data

Full Terms & Conditions of access and use can be found at
https://www.tandfonline.com/action/journalInformation?journalCode=nmcm20

MATHEMATICAL AND COMPUTER MODELLING OF DYNAMICAL SYSTEMS
2026, VOL. 32, NO. 1, 2707718
https://doi.org/10.1080/13873954.2026.2707718

Taylor & Francis
by informa

RESEARCH ARTICLE

OPEN ACCESS Check for updates

An FMI-centric open-source and interoperable framework for ai-driven robotic systems: a model-based methodology for integrating physical and AI models

Sebastian Rojas-Ordoñez[a,b], Mikel Segura[a], Jorge Arboleda[a] and Ekaitz Zulueta[b]

[a]IKERLAN, S. COOP, Spain; [b]University of the Basque Country UPV/EHU, Spain

ABSTRACT

The development of AI-driven cyber-physical robotic systems often depends on proprietary toolchains that limit flexibility and interoperability. This paper presents a modular, open-source, and FMI-centric methodology that integrates OpenModelica-generated Functional Mock-up Units (FMUs), ONNX-based perception pipelines, and ROS 2 middleware into a unified robotic workflow. Beyond software connectivity, the approach makes explicit how perception and control interact through supervisory logic, timing assumptions, and component encapsulation. The methodology is validated on a sign-following mobile robot against an industrial Simulink-based baseline. Both systems achieved 100% task success under nominal conditions. The Simulink baseline was slightly more path-efficient, whereas the FMI-based workflow exhibited substantially lower end-to-end latency. A controlled perception-dropout study further quantified degradation in efficiency, success rate, and latency. A qualitative TurtleBot3 deployment also supports portability beyond simulation. Overall, the results support open, standards-based robotic workflows that are transparent, portable, and reproducible.

ARTICLE HISTORY
Received 9 December 2025
Accepted 17 July 2026

KEYWORDS
Modelica; FMI; Co-simulation; model-based engineering; autonomous robotics; ROS 2; ONNX; simulation-to-hardware portability

1. Introduction

Model-Based Engineering (MBE) has transformed the way engineers and researchers design and develop complex systems (Holtmann et al. 2024). By promoting the reuse of models, tests, and validation artefacts, MBE reduces development effort, shortens design cycles, and improves system quality. Today, both industry and academia rely on MBE to predict system behaviour, reduce dependence on costly physical prototypes, and support increasingly sophisticated robotic applications (Amorim 2019). At the same time, Convolutional Neural Networks (CNNs) are playing an increasingly important role in robotics, especially in perception tasks that support navigation, scene understanding, and decision-making (Javaid et al. 2023).

Despite these advances, many robotic workflows still depend on proprietary tools and closed development ecosystems. Although such tools offer powerful capabilities, their cost, limited interoperability, and vendor lock-in often restrict reuse, transparency, and reproducibility (Haessler et al. 2023). This creates barriers not only for smaller research groups, start-ups, and students, but also for broader collaboration and independent verification. In practice, the reliance on single-vendor toolchains remains in tension with the growing need for portable, reproducible, and cross-domain engineering workflows.

To address these limitations, the research community has developed a set of open standards aimed at interoperability. The Functional Mock-up Interface (FMI), for example, enables control or physics-oriented executable models to be exported as self-contained Functional Mock-up Units (FMUs), decoupling model behaviour from the original simulation environment (Modelica Association 2024). In robotics, the Robot Operating System 2 (ROS 2) has become a widely adopted middleware framework, providing DDS-based communication support for modular and distributed robotic software (Open Robotics 2025). For AI deployment, the Open Neural Network Exchange (ONNX) format provides a

CONTACT Sebastian Rojas-Ordoñez jsrojas@ikerlan.es; sebastianrojas.sero@gmail.com IKERLAN, S. COOP, Spain

© 2026 The Author(s). Published by Informa UK Limited, trading as Taylor & Francis Group.
This is an Open Access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited. The terms on which this article has been published allow the posting of the Accepted Manuscript in a repository by the author(s) or with their consent.

portable mechanism for executing trained models across different platforms and runtimes, independent of the original training framework (ONNX Community 2025).

However, while these standards are individually powerful, their effective integration into a consistent, reproducible, and timing-aware workflow remains a non-trivial engineering problem. In robotic systems, the challenge is not limited to data exchange: it also involves coordinating heterogeneous components that operate under different temporal assumptions, update rates, and failure modes (Kortenkamp et al. 2016). Typical examples include asynchronous perception pipelines, periodically stepped control logic, processed odometry streams, and middleware-level communication delays (Kortenkamp et al. 2016; Betz et al. 2023). Without an explicit treatment of execution semantics, timing assumptions, and data-handling policies, interoperability at the artefact level does not automatically translate into reproducible or behaviourally consistent system-level operation (Gomes et al. 2019; Hansen et al. 2022). As a result, the practical adoption of open standards often remains constrained by the lack of end-to-end methodologies that make execution assumptions and integration semantics explicit.

A central motivation of this work is therefore to show how an open, standards-driven toolchain can be structured as a model-based methodology with explicit execution assumptions. By centreing the workflow on equation-based modelling (Modelica/FMI), robotics middleware (ROS 2), and portable AI inference (ONNX), we show that it is possible to build autonomous robotic systems without relying on proprietary licences. More specifically, this work focuses on the integration of:

- FMU-encapsulated executable components representing control and kinematic processing functions.
- ONNX-deployed perception components representing learned visual inference.

In this context, physical models refer to model-based executable artefacts grounded in kinematics, dynamics, or control logic, whereas AI models refer to learned perception modules deployed for runtime inference. A key principle of the methodology is the reuse of these artefacts across the development lifecycle: the same FMUs, ROS 2 wrappers, and ONNX models are executed under a defined orchestration policy in simulation, hardware-in-the-loop settings, and physical deployment. This portability reduces the need to re-implement functionally equivalent logic across environments and helps narrow the gap between academic prototyping and practical deployment.

Accordingly, the contribution of this paper is not limited to the integration of the FMI, ROS 2, and ONNX, but the definition and validation of an open-source methodology that makes such coupling explicit, portable, and analysable from both modelling and execution perspectives. In particular, the methodology formalises the control layer as a finite-state supervisory structure with defined states, inputs, outputs, and transition logic, while clarifying how model-based and data-driven components interact under heterogeneous timing domains.

This manuscript significantly expands upon our previous conference publication (Rojas-Ordoñez et al. 2025) by upgrading the work from a proof of concept to a methodologically and quantitatively validated study in four main ways:

(1) **A formal execution-oriented open-source methodology** that unifies FMU-based control and processing components, ONNX-based perception, and ROS 2 middleware into a coherent and reproducible workflow.
(2) **An explicit formalisation of the integrated control architecture**, including the specification of the navigation logic as a finite-state supervisory controller with defined states, inputs, outputs, transition logic, and execution assumptions.
(3) **A rigorous quantitative performance analysis** of the integrated system, including task completion, trajectory efficiency, and end-to-end latency metrics.
(4) **A controlled comparative validation and sensitivity-oriented analysis** against an industrial Simulink baseline, including statistically grounded nominal comparison and robustness evaluation under degraded perception availability.

The remainder of this paper is organised as follows. Section 2 reviews the background and related work. Section 3 presents the proposed methodology and architectural formalisation. Section 4 describes the proof

of concept and execution setup. Section 5 reports the quantitative results and discussion. Finally, Section 6 concludes the paper and outlines directions for future work.

2. Background and related work

The development of modern robotic systems increasingly depends on the integration of simulation, control, perception, and software orchestration across heterogeneous components. While recent open-source tools have enabled new approaches for designing and validating complex robotic and cyber-physical systems (Robles et al. 2023; Santos et al. 2024), ensuring reproducible, timing-consistent, and semantically well-defined interaction between these components remains a significant challenge. This section reviews the key technical foundations relevant to our methodology, namely equation-based modelling, MBE, FMI, ROS 2 middleware, and ONNX-based AI deployment. The review is intentionally focused on identifying the integration gap at the intersection of model-based execution, robotic middleware, and deployable learning-based perception.

2.1. Equation-based modelling with modelica

Modelica provides a declarative, equation-based modelling language for describing multi-domain systems through Differential-Algebraic Equations (DAEs). Its acausal formulation decouples the physical representation from solver selection, thereby promoting modularity, reuse, and model portability. Over the past decade, Modelica-based workflows have become increasingly relevant in digital twin initiatives, cyber-physical system design, and cross-domain co-simulation pipelines. Recent work has emphasised symbolic index reduction, equation sorting, and export-oriented compilation strategies that improve simulation robustness and computational efficiency, particularly when models are transformed into FMUs (Moshood et al. 2024; Hatledal 2025).

The open-source OpenModelica software remains a key entry point for generating FMUs in Linux-based environments, supporting both Model Exchange (ME) and Co-simulation (CS) modes (Open Source Modelica Consortium 2024). From the perspective of this paper, the relevance of Modelica is not only its expressiveness for physical-system representation, but also its role as a source of portable executable artefacts that can encapsulate control-oriented and kinematic processing functions within a broader robotic workflow. This capability makes Modelica particularly attractive when reusable model-based components must interact with robotic middleware and AI inference pipelines.

2.2. Simulation in model-based engineering

MBE uses models as central engineering artefacts to support system understanding, design, validation, and lifecycle management (Holtmann et al. 2024). This methodology supports tasks such as requirements capture, system design, simulation-based verification, and code generation, with the aim of reducing development risk and improving traceability (Khandoker et al. 2022). In robotics, MBE has been shown to facilitate iterative refinement, early error detection, and reduced prototyping costs (Sekar and Baras 2022).

A common industrial approach relies on proprietary environments such as MATLAB/Simulink, which provide mature toolchains for control design, simulation, and robotics integration (MathWorks 2024). The Simulink sign-following robot example used as a baseline in this work illustrates a representative architecture in which a Stateflow-based supervisory controller and a perception subsystem interact with a simulated robot in Gazebo through a ROS 2 bridge. While effective from an engineering standpoint, this approach links modelling, integration, and validation to a single vendor ecosystem, thereby reducing portability, limiting independent verification, and constraining reproducibility (Corallo et al. 2022). This limitation is particularly relevant when software artefacts are expected to move across simulation, hardware-in-the-loop, and real robotic deployment without manual re-implementation.

2.3. FMI for model exchange and co-simulation

FMI is an open standard for ME and CS across heterogeneous Modelling and Simulation (M&S) environments. It enables reusable software packaging of executable models as FMUs, which expose model variables, parameters, metadata, and standardised simulation interfaces in a tool-independent manner (Blockwitz et al. 2012). This makes FMI particularly attractive for complex systems in which components from different domains, tools, or organisations must interact through a common integration interface.

Multiple studies have demonstrated the practical value of FMI in cyber-physical and co-simulation settings. For instance, (Oudart et al. 2020) presents a model-based CPS toolchain using FMI to separate physical and logical models while reducing development effort. Similarly, (Shahsavari et al. 2021) introduces MCX as an open-source digital-twin-oriented framework that uses FMUs to encapsulate executable system behaviour in a reproducible manner.

Building on these developments, the integration of FMI with robotics frameworks has emerged as a relevant research direction. One notable contribution is the fmi_adapter package (Lange et al. 2021), which connects FMI with ROS and Gazebo and supports two main integration patterns:

1. **Node-Based Use:** As shown in Figure 1, this approach wraps an FMU inside a ROS node. Subscribers interface with the FMI adaptor, which in turn communicates with the application FMU. This structure simplifies the creation of ROS publishers and subscribers associated with FMU variables.

2. **Library-Based Use:** Figure 2 illustrates a direct integration approach based on the `fmi_adapter::FMIAdapter` C++ class. In this case, an application node interfaces directly with the FMI adaptor components, allowing tighter control over the FMU lifecycle and greater flexibility for custom timing or orchestration policies.

Additionally, (Lange et al. 2021) presented gazebo-fmi plugins for integrating FMUs directly into Gazebo-based simulation environments. In CS settings, these components follow a master–slave execution structure in which the simulator coordinates global timing while individual FMUs evaluate their internal models using their own execution interfaces. The gazebo-fmi suite includes plugins for actuator modelling and fluid-dynamics-related body interactions, illustrating how FMI can extend simulation realism in robotic environments.

Despite these advances, existing FMI–ROS integrations primarily focus on connectivity and software interoperability. They do not, in general, provide a complete methodology for jointly organising model-based control artefacts, AI perception components, and middleware-level execution assumptions within a single reproducible workflow. This distinction is important: interface compatibility alone does not specify

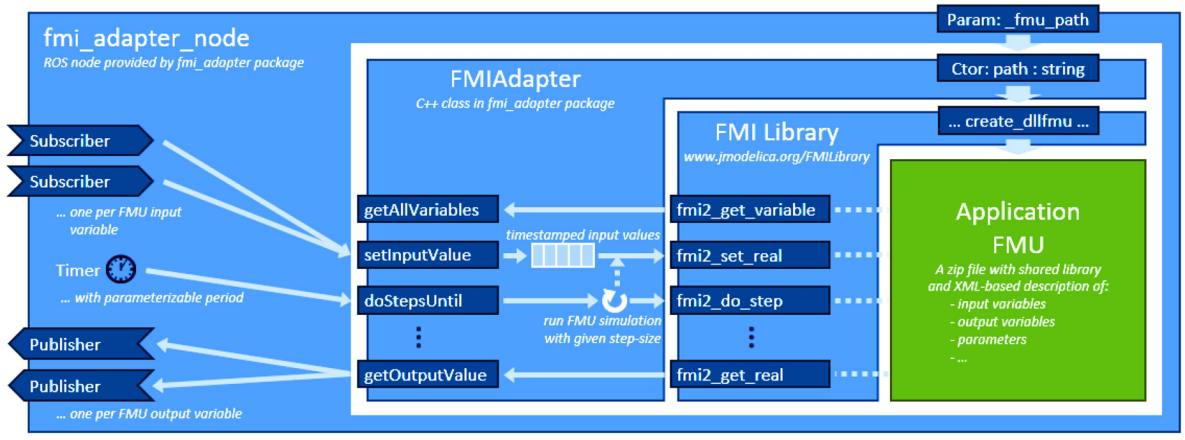


Figure 1. Node-based FMI integration in ROS, adapted from (Lange et al. 2021).

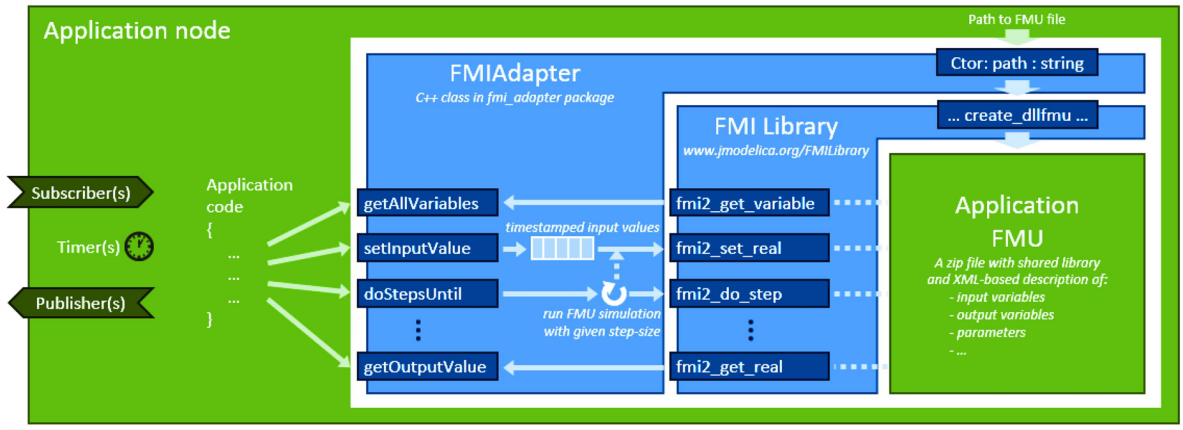


Figure 2. Library-based FMI integration in ROS, adapted from (Lange et al. 2021).

how asynchronous perception updates, FMU stepping, control timers, timeout data, or fallback behaviours should be handled at the system level.

In the scope of this work, we rely on the FMI 2.0 specification, which is widely supported by open-source modelling tools, including OpenModelica. While FMI 3.0 introduces advanced constructs such as clocks and scheduled execution (Modelica Association 2022), robust open-source export workflows for the class of models used in this paper remain less mature. Our use of FMI 2.0 therefore prioritises reproducibility, toolchain stability, and practical accessibility.

2.4. ROS 2 and ONNX for robotics and AI integration

Modern autonomous robotic systems increasingly combine heterogeneous software components, including model-based functions, learned models, sensor interfaces, and actuation interfaces. ROS 2 provides the middleware layer used to compose these components as distributed nodes. Through its publish–subscribe communication model and DDS-based transport, ROS 2 allows independently developed components to exchange typed messages while remaining decoupled at implementation level. This modularity is valuable for robotic applications that must integrate components with different update rates, callback patterns, and deployment requirements (Patel et al. 2022; Open Robotics 2025).

ONNX addresses a complementary problem. It is an open format for representing trained machine-learning models independently of the framework originally used for training. An ONNX artefact can therefore be executed by compatible inference runtimes without embedding training-framework-specific logic in the robotic application. Although visual perception is the use case adopted in this paper, ONNX is not restricted to image processing: it can also represent learned models for state estimation, anomaly detection, classification, prediction, or data-driven decision support (Patrício et al. 2024; ONNX Community 2025).

In the proposed methodology, ROS 2 and ONNX play distinct roles. ROS 2 provides runtime communication and orchestration, whereas ONNX provides a portable representation for AI-based functionality whenever a robotic application includes a learned model. The external interface of each ONNX-based component is application-dependent: the model may consume images, time-series signals, or other tensors and may publish any reduced features required by downstream components. The novelty claimed in this work is therefore not the ONNX format itself, but its systematic composition with FMU-encapsulated model-based components through explicit ROS 2 interfaces and reported execution assumptions.

Existing studies illustrate the practical relevance of ONNX in robotics. For example, learned models have been deployed for real-time control and motion-compensation applications. However, these studies do not address the joint methodological problem of composing ONNX-based AI artefacts with FMU-

based executable components under explicit middleware-level timing and orchestration assumptions (El-Hussieny 2024; Yue 2024).

Current approaches that combine AI perception with robotic control often rely either on proprietary toolchains or on custom, application-specific interface software (Licardo et al. 2024). As a result, the literature still lacks a clearly articulated open-source methodology that simultaneously addresses:

- Portable model-based execution through FMI.
- Deployable AI perception through ONNX.
- Robotic orchestration through ROS 2.

While also making system-level timing and execution assumptions explicit.

In summary, the gap addressed in this paper is not merely the absence of a software connector between FMI and ROS 2 or between ONNX and ROS 2. Rather, the unresolved problem is the lack of an end-to-end, open, and reproducible methodology for combining model-based control artefacts and AI-based perception components into a robotic system whose execution behaviour can be described, implemented, and validated in a transparent manner. Table 1 positions our contribution relative to representative existing approaches. Unlike proprietary workflows, our approach is fully open-source. Compared with FMI–ROS integration tools such as fmi_adapter, our work additionally incorporates ONNX-based perception and targets explicit methodological integration and controlled experimental validation rather than software connectivity alone.

3. Proposed methodology

To address the integration challenges outlined in Section 2, this paper proposes an open-source, execution-oriented model-based methodology for robotic systems based on MBE principles and interoperable standards. Our approach focuses on integrating FMU-encapsulated model-based components and AI-based components deployed as ONNX models within a unified ROS 2 environment. The AI-based component is not restricted to visual perception; its role and interface depend on the robotic application under consideration. Rather than treating interoperability as a purely software-level compatibility problem, the proposed methodology explicitly addresses how heterogeneous components are represented, orchestrated, and executed under different update rates and communication pathways. This framework facilitates the development of complex robotic systems in a modular, portable, and reproducible manner across both simulated environments and physical deployment.

The proposed methodology is guided by three key principles:

- **Interoperability and Open Standards:** The use of open standards (FMI, ONNX, ROS 2) and open-source tools (OpenModelica, Gazebo) supports a modular and scalable architecture, reducing dependence on proprietary ecosystems and improving transparency and reproducibility.
- **Artefact-Level Reuse Across Heterogeneous Domains:** The methodology integrates heterogeneous artefacts, including FMUs for model-based functions and ONNX files for learned functions, by encapsulating them as ROS 2-executable components. This design follows the principle of artefact

Table 1. Positioning of the proposed approach relative to representative related works.

Approach/Reference	Open	FMI	ROS2	ONNX	Exec. Sem.	Baseline Comp.	Robust.	Phys. Val.
Simulink sign-following baseline (MathWorks 2024)	No	Limited	Bridge	Prop.	No	No	No	No
fmi_adapter/Gazebo-FMI (Lange et al. 2021)	Yes	Yes	Yes	No	Limited	No	No	No
MCX digital-twin framework (Shahsavari et al. 2021)	Yes	Yes	No	No	Partial	No	No	No
FMI-based CPS toolchain (Oudart et al. 2020)	Partial	Yes	No	No	Partial	No	No	No
ONNX robotic control/perception studies (El-Hussieny 2024; Yue 2024)	Partial	No	No	Yes	Limited	No	No	No
Custom AI–robot integration works (Licardo et al. 2024)	Partial	No	Partial	Yes	Limited	No	No	Partial
This Work	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**

Note: The comparison reflects only capabilities explicitly reported or directly demonstrated in the cited works. "Partial" indicates that the aspect is present only indirectly, in a restricted form, or without end-to-end methodological treatment.

reuse: previously developed and validated models are incorporated into the robotic system without manual re-implementation of their functional core. This yields several practical advantages:

- **Reduced Development Effort:** It reduces development time and implementation overhead by enabling the reuse of validated artefacts across design and deployment stages.
- **Enhanced Modularity and Maintainability:** Individual modules (e.g. a controller FMU or a perception model) can be updated, tested, or replaced independently without requiring changes to the full system.
- **Ecosystem Compatibility:** Since many robotic systems are structured around ROS 2, new functionalities can be added as interoperable components within an existing communication graph.

- **Explicit Separation Between Functional Modelling and Runtime Orchestration:** Our design adopts a modular architecture in which independent subsystems are wrapped within ROS 2 nodes, ROS 2 provides the communication substrate, and robot or simulator interfaces provide the connection to the target platform. This separation allows the functional behaviour of model-based and AI-based artefacts to remain distinct from the middleware level orchestration required for execution, buffering, and message transport. As a result, the proposed methodology is adaptable to a wide range of robotic applications while remaining analysable from both a modelling and a systems-integration perspective.

3.1. General methodological workflow

The proposed methodology can be applied beyond the specific mobile-robot case study through the following step-by-step procedure:

(1) **Identify functional roles:** Decompose the robotic application into model-based functions, AI-based functions when required, and robot or simulator interfaces.
(2) **Package reusable executable artefacts:** Export model-based functions as FMUs and learned functions as ONNX artefacts so that their functional cores remain independent of the middleware integration code.
(3) **Define ROS 2 data contracts:** Specify the topics, message types, units, tensor transformations, and reduced features exchanged by the components.
(4) **Implement wrapper-level orchestration:** Encapsulate the artefacts in ROS 2 nodes and define their triggering rules, stepping periods, buffering policies, and data-retention assumptions.
(5) **Define application-specific supervisory and fallback behaviour:** Select the controller states, transition conditions, and safe responses that are appropriate for the target robotic platform and operational context.
(6) **Validate incrementally:** Verify standalone artefacts, confirm numerical integrity after ROS 2 integration, and finally evaluate task-level behaviour under nominal and degraded conditions.

This workflow separates the reusable integration procedure from the application-specific artefacts selected for a particular robotic system.

3.2. System architecture

The core of the methodology is a distributed architecture in which reusable model-based components, AI-based components when required, and robot or simulator interfaces are encapsulated in dedicated ROS 2 nodes or wrappers. This separation of concerns improves modularity, supports independent testing of subsystems, and simplifies deployment across simulation and hardware-oriented settings. Figure 3 illustrates this general software architecture.

The three main architectural roles are:

(1) **Model-Based Components:** FMU-encapsulated functions representing physical models, signal-processing functions, control logic, or other reusable equation-based artefacts.
(2) **AI-Based Components:** ONNX-deployed learned functions whose inputs and outputs are defined according to the application, such as perception, estimation, classification, or prediction.

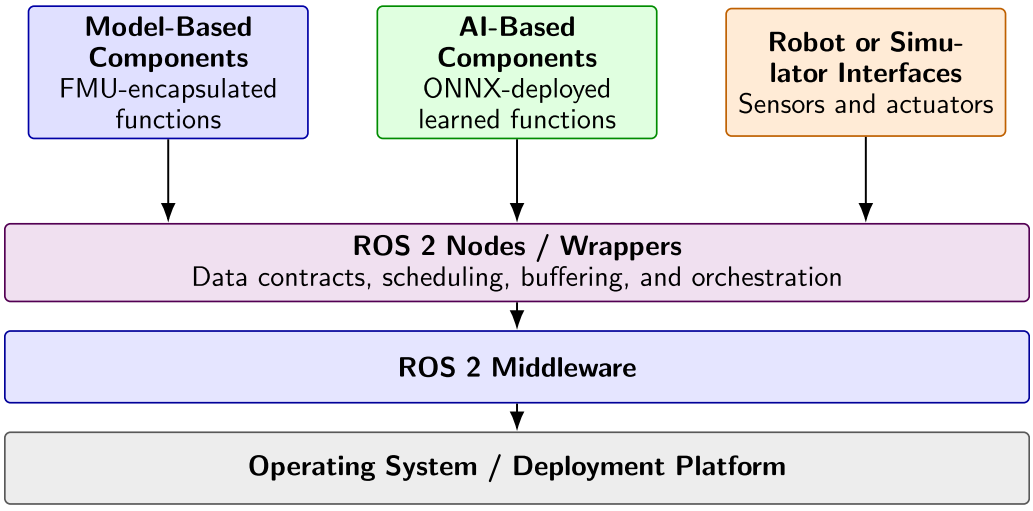


Figure 3. General software architecture of the proposed methodology, separating reusable FMU-based components, ONNX-based AI components, ROS 2 orchestration, and robot or simulator interfaces.

(3) **Robot or Simulator Interfaces:** Components that connect the reusable artefacts to sensors, actuators, and simulation or hardware-specific communication endpoints.

These roles are connected through ROS 2 nodes and wrapper-level orchestration. Their concrete instantiation is application-dependent. In the proof of concept presented in Section 4, the roles are instantiated as an ONNX-based visual-perception node, an FMU-based odometry-processing node, and an FMU-based sign-tracking controller connected to Gazebo and subsequently reused on a physical TurtleBot3 platform.

The integration methodology is intended to support heterogeneous robotic platforms, including mobile robots, robotic arms, and aerial systems, provided that suitable components and interfaces are defined for the target application. Accordingly, the formalisation below is deliberately application-independent: signal dimensions, controller modes, transition conditions, and fallback actions are left to the concrete instantiation. Their case-study realisation for the sign-guided mobile-robot proof of concept is deferred to Section 4.2.3, after the application scenario has been introduced.

3.3. Formalisation of the integrated control layer

To move beyond an implementation-level description, the methodology formalises the integrated control layer as a discrete-time supervisory structure coupled to heterogeneous model-based components and, when present, AI-based components. The purpose of this formalisation is twofold:

(1) Make the model-based control behaviour portable and reproducible when encapsulated as an FMU.
(2) Clarify the assumptions governing interactions between model-based components and any AI-based components within the ROS 2 execution environment.

Let the supervisory mode at control step k be denoted by

$$s_k \in S = \{s^{(1)}, \ldots, s^{(N_s)}\}, \tag{1}$$

where $\mathcal{S}$ is the finite set of application-defined operating modes. When the controller must retain information between evaluations, let $m_k \in M$ collect its internal memory variables. The complete supervisory state is then

$$q_k = (s_k, m_k) \in Q = \mathcal{S} \times \mathcal{M}. \tag{2}$$

This distinction allows the formal model to represent both the visible operating mode and application-specific persistence, such as latched decisions, counters, or stored references. For a memoryless supervisor, $\mathcal{M}$ reduces to a singleton and q_k is equivalent to s_k.

The processed input and control-output vectors belong to the application-defined spaces

$$z_k \in Z \subseteq \mathbb{R}^{n_z}, \tag{3}$$

$$u_k \in \mathcal{U} \subseteq \mathbb{R}^{n_u}, \tag{4}$$

where z_k collects the processed state, learned features, estimates, and status information required by the supervisory layer, while u_k contains the commands or references exposed to the target platform. The dimensions n_z and n_u, the ordering and units of their elements, and their ROS 2 representations form part of the application data contract rather than the generic control formalism.

At the methodological level, the controller is described by mappings $F\colon \mathcal{Q} \times \mathcal{Z} \times \mathcal{T} \to \mathcal{Q}$ and $G\colon \mathcal{Q} \times \mathcal{Z} \to \mathcal{U}$, with the discrete state-update and output relations

$$q_{k+1} = F(q_k, z_k, \tau_k), \tag{5}$$

$$u_k = G(q_k, z_k), \tag{6}$$

where $\tau_k \in \mathcal{T}$ denotes the execution and timing context available to the supervisor, such as elapsed durations or freshness information when the data contract provides it. Equation (5) updates both the operating mode and its internal memory, whereas (6) generates the platform-facing command. The sets $\mathcal{S}$, $\mathcal{M}$, $\mathcal{Z}$, $\mathcal{U}$, and $\mathcal{T}$, together with the mappings F and G, form an application-independent template that is specialised for the target system.

The formulation in (3)–(6) does not require synchronous updates across all input channels. A wrapper may construct z_k from the most recent values available at each control step, provided that its triggering, holding, and freshness policies are stated explicitly. The proof-of-concept instantiation and its concrete execution semantics are presented in Sections 4.2.3 and 4.4, respectively.

3.3.1. Generic finite-state supervisory realisation

A finite-state implementation realises the mode component of (5) through application-defined transition guards. For a current mode $s^{(i)}$, each admissible successor $s^{(j)}$ is associated with a guard evaluated from z_k, m_k, and τ_k. The mapping F selects the next mode and updates any persistent variables, while G associates the current complete supervisory state with a command in $\mathcal{U}$.

This generic representation does not prescribe the number or names of the modes, the physical meaning of the inputs and outputs, or the guard thresholds. Instead, it distinguishes the information needed to describe a concrete realisation: the mode set, persistent variables, transition conditions, output actions, initialisation, and execution assumptions. The finite-state realisation used in the proof of concept is specified only after the navigation task and platform have been introduced in Section 4.2.3.

3.3.2. Application-dependent fallback specification

Fallback behaviour is represented through designated guards, modes, and output actions within F and G. Its trigger may be an invalid input, a missing or stale update, an operating-limit violation, or another application-defined condition. The corresponding response is platform dependent: examples include holding a previous command, entering a recovery mode, commanding zero motion, hovering, controlled landing, or moving a manipulator to a predefined configuration.

The data contract must also distinguish between an explicitly invalid sample and the absence of a fresh sample. A supervisor can react to stale data only if freshness is observable through a timestamp, validity indicator, elapsed-time value, or wrapper-level timeout. If the orchestration layer exposes only retained values, complete stream interruption cannot be distinguished from repeated reuse of the last sample. For this reason, each application must report both its fallback logic and the message-retention policy that

determines which conditions the controller can actually observe. The concrete recovery and stopping behaviour of the proof of concept is described in Section 4.2.3.

3.4. Data-interface specification

The communication between components is governed by explicit data contracts. At minimum, each contract identifies the source and destination roles, semantic quantity, data type and shape, element ordering, units and reference frame, and the middleware endpoint used to transport it. These properties determine how FMU variables, ONNX tensors or reduced features when present, and robot-facing messages are mapped without requiring either functional artefact to depend on the other's internal representation.

A data contract must also specify temporal semantics that cannot be inferred from topic names or message types alone. These include the production and consumption triggers, expected update rates, initialisation, buffering or sample-hold policy, validity and freshness representation, and the response to unavailable data. The signal-level and temporal parts of the contract jointly determine the observable coupling between otherwise independent components.

The concrete ROS 2 topics, message types, dimensions, and units used in the proof of concept are reported in Section 4.3, after its perception, odometry, and control components have been introduced.

4. Proof of concept: indoor navigation with traffic signals

To validate the proposed methodology, we instantiate it in a robotic navigation scenario in which a mobile robot moves from a starting point to a destination by following traffic signals. This use case was adapted from a MathWorks Simulink example originally designed to demonstrate communication between Gazebo and Simulink (MathWorks 2024). The purpose of this instantiation is not merely to reproduce the behaviour of the proprietary example, but to evaluate whether an open-source methodology based on FMI, ONNX, and ROS 2 can provide a functionally equivalent and experimentally analysable alternative under controlled conditions.

A key aspect of this work was addressing the interoperability challenges that arise when moving from a proprietary single-vendor environment to a multi-tool open-source one. Because our development environment was Linux-based, the original Windows-oriented Simulink artefacts could not be exported directly as FMUs. Consequently, the odometry and control logic modules were reimplemented in OpenModelica and exported as FMUs from there. Similarly, the proprietary Simulink image-processing blocks were replaced by a deep-learning-based perception pipeline using YOLOv8, which was then exported to ONNX for deployment. The original Gazebo simulation environment was reused as a common validation world, thereby preserving the task structure and enabling a controlled comparison across toolchains.

4.1. Application scenario and platform

Figure 4 shows the simulated Gazebo environment used in this work. In this scenario, the mobile robot must navigate from point A to point B while following visual traffic signals. The yellow dashed line represents the expected path that the robot should follow according to the signs placed throughout the environment.

The robot navigation task is guided by the visual signals shown in Figure 5. These signs provide the commands that determine the robot behaviour at key points along the route.

The TurtleBot3 Burger is a differential-drive robot. Its kinematics can be described by the standard unicycle model, where the robot's state is its 2D pose (x, y, θ). The forward kinematics, which relate the robot's body-frame velocities (v, ω) to the rate of change of its pose in the world frame, are described by the standard unicycle model (Siegwart et al. 2011):

$$\dot{x} = v\cos\theta, \quad \dot{y} = v\sin\theta, \quad \dot{\theta} = \omega \tag{7}$$

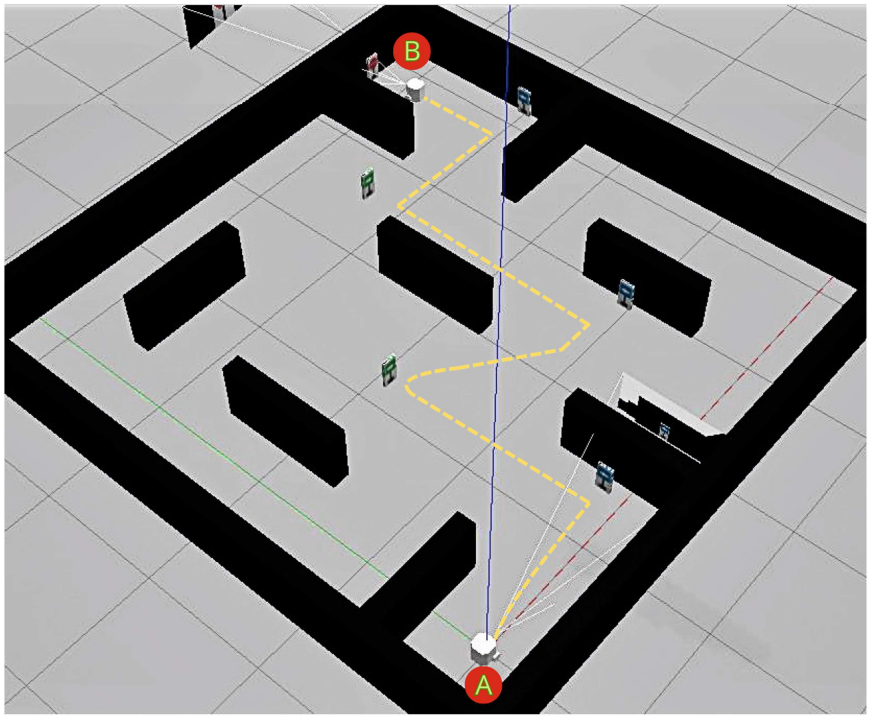


Figure 4. Simulated Gazebo environment showing the navigation path and the traffic signs that guide the robot behaviour.

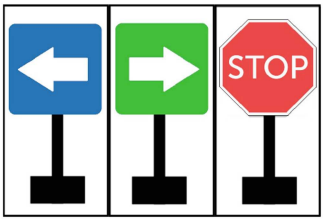


Figure 5. Traffic signals used for robot navigation: left-turn arrow, right-turn arrow, and stop sign.

In Equation (7), the linear velocity v and angular velocity ω are derived from the individual wheel angular velocities (ω_R, ω_L) as follows in Equation (8):

$$v = \frac{r}{2}(\omega_R + \omega_L), \quad \omega = \frac{r}{2b}(\omega_R - \omega_L) \tag{8}$$

where r is the wheel radius and $2b$ is the wheel separation (track). These equations are not used as a high-fidelity plant model inside the controller FMU; rather, they define the planar kinematic interpretation that motivates the odometry-processing and control abstractions adopted in this proof of concept.

4.2. Component instantiation

Following the proposed methodology, the core functionalities of the robotic system were instantiated as three ROS 2 runtime components corresponding to perception, odometry processing, and control.

4.2.1. Traffic sign detection ONNX

A robust perception subsystem is required to support traffic-sign-guided navigation. For this purpose, we implemented an object-detection-based perception component and deployed the resulting trained model through ONNX.

The process began with the creation of a custom dataset and the fine-tuning of a pretrained YOLOv8-s model using transfer learning. The complete training and export protocol is summarised in Table 2. The best-performing checkpoint was then exported to ONNX for framework-independent deployment.

At the methodological level, an ONNX-based subsystem is treated as an executable inference artefact that exposes an application-defined reduced interface to the rest of the robotic architecture. In the present case study, this role is instantiated as a perception subsystem. This abstraction is deliberate: rather than coupling the control layer to full detector tensors or detector-specific internals, the perception node publishes only the compact descriptors required by the controller. In this sense, the perception pipeline is not merely a software module, but a deployable functional component with a defined external contract.

Figure 6 illustrates the case-study-specific instantiation of this general ONNX-based role. It is not a mandatory interface for every application using the methodology. In this experiment, the component acts as a black box that receives an image tensor and produces the perception features required by the control FMU. Other applications may use different ONNX inputs and outputs.

The perception component is implemented as a Python-based ROS 2 node using `rclpy`, `cv_bridge`, and ONNX Runtime. The node subscribes to `/camera/image_raw`, converts each incoming image into an RGB tensor, resizes it to 640 × 480 when required, casts it to float32, and adds the batch dimension before inference. Inference is callback-driven rather than timer-driven: each received image triggers a complete preprocessing, inference, decoding, and publication cycle.

The ONNX model output is interpreted as a structured feature vector in which the first half of the output corresponds to horizontal blob-position features and the second half corresponds to blob-size features. In the present task, these features are organised as three class-wise channels, corresponding to the three sign categories considered in the navigation scenario.

These values are published to the `/onnx_blobX` and `/onnx_blobSize` topics as three-element vectors, allowing the controller FMU to receive class-wise perception descriptors rather than a single undifferentiated blob measurement. Because these outputs are generated only when image callbacks occur, the perception stream is asynchronous with respect to the periodic FMU-based control loop. Consequently, the control wrapper consumes these features using a latest-sample, sample-and-hold policy, as described in Section 4.4.

This design choice decouples the detector implementation from the control layer while maintaining a compact and reproducible perception–control interface. The final integration of this node within the ROS 2 ecosystem is verified at runtime using the communication graph shown in Figure 7.

4.2.2. Odometry component (FMU)

The odometry component is responsible for transforming the simulator-provided pose representation into the planar pose variables required by the navigation controller. This functionality is encapsulated in an FMI 2.0 Co-Simulation FMU generated from an OpenModelica model named ReadOdometry.

Table 2. Consolidated training and deployment protocol for the YOLOv8 perception model.

Parameter	Specification
Model Architecture	
Backbone/Head	YOLOv8-s (pretrained on COCO)
Image Size	640 × 640 (with letterbox resize)
Training Protocol	
Batch Size	16
Epochs	100 (with early stopping, patience = 20)
Optimiser	AdamW
Initial Learning Rate	3×10^{-4} (with cosine decay)
Weight Decay	1×10^{-4}
Data Augmentations	Horizontal Flip ($p = 0.5$), Rotations (±10°), HSV Jitter, Brightness/Contrast, Mosaic ($p = 0.2$), MixUp ($p = 0.1$)
Validation and Export	
Selection Criterion	Best $mAP_{50:95}$ on validation set
Export Format	ONNX (opset ≥ 12) with post-export numerical validation against PyTorch
Deployment Parameters (ROS 2 Node)	
Confidence Threshold	0.25
NMS IoU Threshold	0.5
Inference Provider	CPU/CUDA/TensorRT

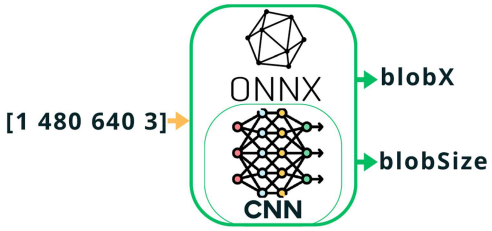


Figure 6. Case-study-specific instantiation of the ONNX-based AI component as a visual-perception subsystem, showing the reduced interface exposed to the rest of the system.



Figure 7. Runtime communication graph generated by `rqt_graph`, showing the perception node subscribing to `/camera/image_raw` and publishing the extracted perception features.

The FMU receives the Cartesian position inputs x and y, together with the quaternion components qw, qx, qy, and qz, and produces the planar outputs posX, posY, and theta. According to the FMU model description, the translational outputs depend directly on the Cartesian position inputs, while the heading output depends on the quaternion channels. In this way, the component acts as a model-based state-transformation artefact that normalises the simulator odometry into a representation directly consumable by the supervisory controller.

Its main role is therefore to convert the 6-DoF odometry information from the `/odom` topic into the reduced planar pose (x, y, θ). The simulator provides Cartesian position (x, y) and orientation in quaternion form (q_w, q_x, q_y, q_z). While the position coordinates are passed through directly, the core task of the FMU is to convert the quaternion into a single 2D orientation angle-the yaw angle, θ. This is achieved using the standard Euler angle conversion formula:

$$\theta = atan2\left(2(q_w q_z + q_x q_y),\ 1 - 2\left(q_y^2 + q_z^2\right)\right) \tag{9}$$

This transformation, shown in Equation (9), isolates the rotation around the vertical axis, which is the relevant orientation for a differential-drive robot operating on a plane. Figure 8 illustrates the I/O structure of this FMU, showing the six inputs and the three resulting outputs that describe the robot's complete 2D pose.

This FMU is wrapped in a C++ ROS 2 node named `fmu_odometry_node`. The wrapper subscribes to `/odom`, stores the most recent odometry message, writes the corresponding values into the FMU inputs, advances the FMU through the adaptor interface, and publishes the resulting planar pose to `/fmu_odom_out` as a `geometry_msgs/Pose2D` message. As in the control wrapper, this execution is driven by an external periodic timer rather than by asynchronous FMU self-execution.

In the implemented system, the odometry wrapper also uses a periodic stepping policy with a 10 ms wall timer. If no odometry message has yet been received, the FMU is not advanced and no processed pose is published. Once the first odometry message arrives, the latest received sample is reused at each wrapper step until a new message becomes available. Accordingly, the odometry interface also follows a latest-sample, sample-and-hold integration policy at the ROS 2 orchestration layer.

From the perspective of the overall methodology, this component illustrates that FMUs can be used not only for control synthesis but also for intermediate state processing and interface normalisation between

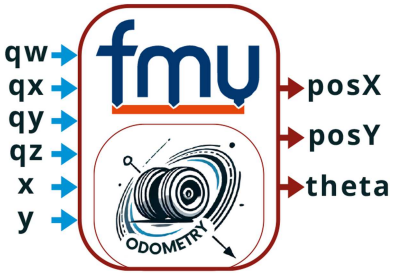


Figure 8. Input–output structure of the odometry FMU, showing quaternion and position inputs and the resulting planar pose outputs.



Figure 9. ROS 2 integration of the odometry FMU, showing subscription to `/odom` and publication to `/fmu_odom_out`.

simulator-provided data structures and controller-oriented representations. As verified by the runtime communication graph in Figure 9, this node correctly subscribes to the simulator's `/odom` topic and publishes the processed 2D pose to the custom `/fmu_odom_out` topic for consumption by the control logic.

4.2.3. Control component (FMU)

The control component is the central supervisory element of the system and is encapsulated in an FMU generated from an OpenModelica model named `SignTrackingLogic`. The exported artefact is an FMI 2.0 Co-Simulation FMU, which is integrated into ROS 2 through a dedicated wrapper node based on the fmi_adapter library. This component provides the application-specific instantiation of the generic integrated control layer defined in Section 3.3; its state-update and output mappings are denoted by F_{nav} and G_{nav}, respectively.

For this navigation application, the generic processed input z_k is instantiated as

$$z_k^{\text{nav}} = \left[x_k \quad y_k \quad \theta_k \quad (\mathbf{b}_k^x)^{\mathrm{T}} \quad (\mathbf{b}_k^s)^{\mathrm{T}} \right]^{\mathrm{T}} \in \mathbb{R}^9, \tag{10}$$

where (x_k, y_k, θ_k) is the processed planar pose, $\mathbf{b}_k^x \in \mathbb{R}^3$ is the vector of class-wise horizontal blob positions, and $\mathbf{b}_k^s \in \mathbb{R}^3$ is the vector of class-wise blob sizes. The generic control output is instantiated as

$$u_k^{\text{nav}} = [v_k \quad \omega_k]^{\mathrm{T}} \in \mathbb{R}^2, \tag{11}$$

where v_k and ω_k are the commanded linear and angular velocities. Thus, the application sets $n_z = 9$ and $n_u = 2$ in Equations (3) and (4). The Boolean stopRobot value exposed by the FMU is an auxiliary supervisory output and is not part of the platform command vector u_k^{nav}. The wrapper evaluates F_{nav} and G_{nav} every $T_s = 10$ ms; no per-message freshness quantity is supplied to the controller. Section 4.4 gives the complete runtime semantics.

Unlike a purely combinational controller, the `SignTrackingLogic` FMU implements stateful supervisory logic through discrete internal variables. In the notation of Equation (2), m_k records whether a valid sign has been detected, which sign class is currently dominant, the selected blob location, whether a turn-in-place manoeuvre is active, its stored direction, the current scan direction, and whether a persistent stop condition has been reached. The resulting behaviour is event-driven and history dependent rather than a function of the current input sample alone.

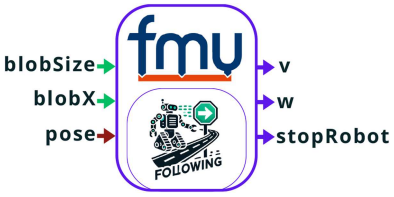


Figure 10. Input–output structure of the `SignTrackingLogic` FMU, showing class-wise perception inputs, pose inputs, and control-related outputs.

The FMU receives three class-wise blob-size inputs (`blobSize[1..3]`), three class-wise horizontal blob-position inputs (`blobX[1..3]`), and a three-element robot pose vector (`pose[1..3]`). It produces the commanded linear velocity v, the commanded angular velocity w, and the Boolean stop flag stopRobot. Figure 10 illustrates this I/O configuration.

A valid detection is declared when at least one blob-size channel exceeds the configured detection threshold. Outside turn-in-place mode, the dominant sign index is selected as the class with maximum blob size, and the corresponding horizontal image position is used for heading correction. While a turn-in-place manoeuvre is active, the previously selected sign index and blob location are retained until the manoeuvre is completed. The controller then combines:

- Scan behaviour when no valid blob is available.
- Image-based heading correction when the selected blob is laterally displaced.
- Longitudinal motion regulation based on perceived blob size.
- Sign-dependent manoeuvre selection in the intermediate proximity range.

In the current implementation, red triggers a persistent stop condition, while green and blue trigger stored turn-in-place manoeuvres in opposite directions.

Finite-state interpretation For exposition, the application-specific mode set is

$$\mathcal{S}_{\text{nav}} = \{\text{SEARCHING},\ \text{APPROACHING},\ \text{EXECUTING},\ \text{SAFE_STOP}\}. \tag{12}$$

For this mode-level interpretation, $\mathcal{S} = \mathcal{S}_{\text{nav}}$ in Equation (1), with the following meanings:

- **SEARCHING:** When no valid sign is available, the controller applies scan behaviour until a sufficiently large class-wise blob is detected.
- **APPROACHING:** Once a valid blob is available and no turn-in-place manoeuvre is active, blob size and horizontal image position regulate forward motion and heading correction.
- **EXECUTING:** When the selected blob reaches the intermediate decision range, the controller triggers the sign-dependent stop or turn-in-place behaviour.
- **SAFE_STOP:** Once the persistent stopping condition has been reached, the controller commands terminal zero motion.

Table 3 relates these application-specific modes to their inputs, output roles, and representative guards. The FMU's internal variables refine this mode-level interpretation through the memory component m_k of the complete state q_k.

Together with the persistent variables described above, the executable FMU rules specialise the generic mappings F and G as F_{nav} and G_{nav}. Table 3 is a mode-level summary of those rules rather than an exhaustive restatement of the underlying Modelica implementation.

Recovery and terminal behaviour The controller distinguishes recovery from the nominal terminal action:

(1) **Search-Oriented Recovery:** This response instantiates the fallback specification from Section 3.3.2. When the perception node publishes values that contain no valid sign, the controller applies scan behaviour rather than continuing nominal sign-guided motion.

Table 3. Supervisory FSM summary for the sign-tracking controller.

Mode	Main Inputs	Nominal Output Role	Representative Transition Condition
SEARCHING	Class-wise blob sizes	Exploratory scan/no-target behaviour	A valid blob exceeds the detection threshold
APPROACHING	Blob position, blob size, pose	Heading correction + forward/backward regulation	Blob enters decision range
EXECUTING	Dominant sign channel, stored manoeuvre direction	Stop or turn-in-place manoeuvre	Manoeuvre completion or persistent stop activation
SAFE_STOP	Internal stop condition	Zero motion command	Terminal stop condition reached



Figure 11. Runtime communication graph of the `SignTrackingLogic` node, showing subscriptions to perception and odometry topics and publication to `/cmd_vel`.

(2) **Nominal Terminal Stop:** When the task's stop-sign condition is reached, the controller enters SAFE_STOP and commands $v_k = 0$ and $\omega_k = 0$.

These responses are encoded inside the FMU rather than in an external middleware timeout manager. Because no freshness quantity is supplied, recovery can be triggered by a published invalid or weak detection but not by complete stream interruption after a valid sample. Section 4.4 reports the corresponding sample-hold limitation explicitly.

This `SignTrackingLogic` FMU is wrapped in a C++ ROS 2 node named `fmu_sign_tracking_node`. The node manages topic subscriptions, updates the FMU inputs, advances the FMU through the adaptor interface, and publishes the resulting velocity command to `/cmd_vel`. From the methodological perspective, this node is not merely a transport wrapper; it acts as the practical orchestration layer that connects asynchronous ROS 2 inputs with periodic FMU-based control evaluation. The successful integration and correct communication pathways are verified at runtime using `rqt_graph`, as shown in Figure 11.

4.3. Application-specific data interfaces

The generic data-interface specification in Section 3.4 is instantiated here through the ROS 2 topics and message types that couple the proof-of-concept components. The topics `/onnx_blobX` and `/onnx_blobSize` transport three-element vectors in the detector's fixed class-index order (classes 0, 1, and 2), allowing the `SignTrackingLogic` FMU to select the dominant sign internally.

Similarly, `/fmu_odom_out` is not a direct simulator output but the result of the FMU-based normalisation stage that converts raw odometry into the planar pose required by the controller. Table 4 records the source and consumer roles, endpoints, payload fields, ordering, units, and reference-frame interpretation of the application data contract. No separate validity or freshness field is transported: detection validity is inferred from the blob-size channels, whereas message freshness is not observable to the FMU. The associated triggering, initialisation, holding, and stale-data policies are reported in Section 4.4.

4.4. Execution semantics and runtime assumptions

Because reproducibility in heterogeneous robotic workflows depends not only on software connectivity but also on temporal coordination, the present proof of concept makes the main execution assumptions

Table 4. Proof-of-concept data interfaces, payload mappings, units, and reference-frame interpretations.

Signal	Source	Consumer	Topic/Type	Payload Used/Ordering	Units/Frame
Camera Image	Gazebo	Perception Node	/camera/image_raw sensor_msgs/Image	Image payload; RGB 640 × 480 × 3 after preprocessing	pixels; camera frame
Raw Odometry	Gazebo	Odometry Node	/odom nav_msgs/Odometry	Position (x, y); quaternion (q_w, q_x, q_y, q_z)	m; unit quaternion; simulator frame
Sign Size Vector	Perception Node	Control Node	/onnx_blobSize geometry_msgs/Vector3	x/y/z: detector class indices 0/1/2	pixels 2
Sign Position Vector	Perception Node	Control Node	/onnx_blobX geometry_msgs/Vector3	x/y/z: detector class indices 0/1/2	pixels; image frame
Processed Pose	Odometry Node	Control Node	/fmu_odom_out geometry_msgs/Pose2D	x, y, theta	m, rad; simulator planar frame
Velocity Cmd	Control Logic Node	Robot/Gazebo Interface	/cmd_vel geometry_msgs/Twist	linear.x = v_k; angular.z = ω_k	m/s, rad/s; robot body frame

explicit. In the evaluated setup, the system operates on a single machine within a ROS 2/Gazebo environment. The perception, odometry, and control components are implemented as separate ROS 2 nodes, but they execute within the same host platform, thereby avoiding additional network-induced uncertainty associated with distributed deployment.

Both the ReadOdometry and SignTrackingLogic components are deployed as FMI 2.0 Co-Simulation FMUs and externally orchestrated by dedicated ROS 2 wrapper nodes. In the implemented system, fmu_odometry_node and fmu_sign_tracking_node act as the practical orchestration layer for the FMU-based pipeline. Each wrapper advances its corresponding FMU periodically every 10 ms through a wall timer, yielding an effective stepping period of T_s = 10ms (100 Hz) for both odometry processing and control evaluation. At each timer event, the wrapper writes the current input values into the FMU through the adaptor interface and then advances the FMU using doStepsUntil(current_time).

Perception and raw odometry updates arrive asynchronously through ROS 2 subscriptions. The odometry wrapper subscribes to /odom, stores the latest Cartesian position and quaternion orientation, and periodically converts them into the planar pose representation required by the controller. The perception node subscribes to /camera/image_raw and runs ONNX inference in an event-driven manner, i.e. each incoming image triggers preprocessing, inference, output decoding, and publication of the resulting features. The perception outputs are published as three-element vectors on /onnx_blobX and /onnx_blobSize, representing class-wise horizontal position and size descriptors.

At the orchestration layer, the wrappers do not perform interpolation or explicit buffering of message histories. Instead, they follow a latest-sample, sample-and-hold policy: at each periodic step, the most recently received values are written into the FMU inputs and reused until new values become available. In the control wrapper, if a given input stream has not yet produced any message, zero-valued defaults are applied. After the first reception, the last valid sample is retained across control cycles because the reception flags are not reset after each step. Thus, the practical missing-data behaviour is based on default initialisation followed by last-value retention. This policy bridges asynchronous update rates but does not by itself communicate sample freshness to the FMU.

The current wrapper-level implementation does not perform interpolation between samples and does not maintain an explicit time-since-last-message timeout outside the FMU. Therefore, stale-data handling at the orchestration level is limited to last-value retention. In particular, after a valid perception update, a complete interruption of the perception stream is not distinguishable from repeated reuse of the last sample. This limitation is reported explicitly because validity flags, timestamps, or timeout-based fallback policies should be added when the target application requires freshness-aware behaviour.

The implementation uses the ROS 2 default publisher/subscriber configuration with queue depth 10 for the relevant topics. Since no explicit custom QoS profile was defined in the present wrappers, the communication assumptions reported here correspond to the default ROS 2 setup used by the middleware in this deployment.

It is important to note that both exported FMUs declare themselves as FMI 2.0 Co-Simulation artefacts with support for variable communication step sizes and input interpolation at the FMU

interface level. However, the practical execution semantics of the complete robotic system are determined by the external ROS 2 orchestration policy described above. Consequently, the effective system behaviour depends on both the FMU capabilities and the wrapper-level scheduling and data-handling logic.

4.5. System integration and experimental protocol

The three components described above were integrated into a complete robotic system using ROS 2 as communication middleware. The final integration phase unifies the FMU-based components and the ONNX-based perception module into a common execution graph. The resulting communication structure is shown in Figure 12.

To evaluate the proposed methodology, we adopted a controlled protocol composed of two complementary parts:

(1) A nominal comparison against an industrial baseline.
(2) A robustness-oriented sensitivity analysis under controlled perception dropout.

This structure allows the manuscript to evaluate both nominal competitiveness and degradation behaviour under reduced perception availability.

4.5.1. Baseline description

The baseline for comparison is the MathWorks "Sign-Following Robot with ROS 2" example (MathWorks 2024). This reference implementation controls a simulated TurtleBot3 in Gazebo from MATLAB/Simulink through a ROS 2 bridge and includes proprietary blocks for sign detection and supervisory control. Our approach replaces these proprietary elements with components based on FMI and ONNX while preserving the same task objective and simulation environment.

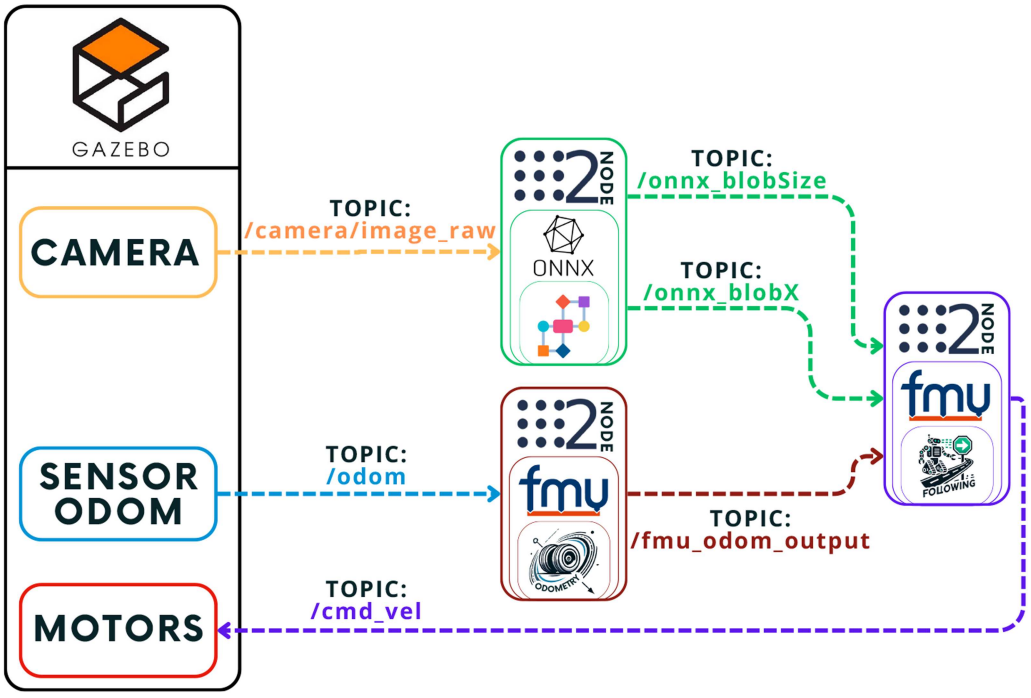


Figure 12. Integrated communication structure of the complete system, showing the interaction between simulation, perception, FMU-based processing, and control publication.

4.5.2. Comparison protocol

To ensure a controlled comparison, the following experimental conditions were aligned between the proposed system and the baseline:

- **Identical Environment and Platform:** Both systems were executed in the same Gazebo world, using the same TurtleBot3 model, starting pose, camera configuration, and host hardware.
- **Identical Task Objective:** In both cases, the robot was required to complete the same sign-guided navigation mission from the same initial location.
- **Comparable Communication Interfaces:** Both systems relied on the same core ROS 2 topics for image input, odometry, and velocity command (`/camera/image_raw`, `/odom`, `/cmd_vel`), with matched motion limits.

  This alignment reduces confounding factors and helps ensure that observed differences are primarily attributable to the respective software methodologies and execution pipelines rather than to differences in environment configuration.

4.5.3. Quantitative performance metrics

Both systems were evaluated using objective task-level and timing-level metrics. In the nominal navigation task, both our system and the Simulink baseline achieved a 100% task completion rate across the evaluated runs. Given this ceiling effect in success rate, the nominal comparison focuses on secondary metrics that better capture execution quality and responsiveness:

- **Trajectory Efficiency (η):** the ratio between the optimal path length L_{opt} and the actual path length L_{real} travelled by the robot, which quantifies path-following efficiency.
- **End-to-End Latency:** the elapsed time between image acquisition and publication of the corresponding velocity command. Latency is reported through percentile-based statistics (p50, p95, and p99) and visualised in the results section through distribution plots across runs.

To complement the nominal comparison, the sensitivity study under perception dropout evaluates three additional system-level indicators:

(1) The evolution of trajectory efficiency under degraded perception availability.
(2) The corresponding success rate.
(3) The increase in latency tails as perception dropout increases.

4.5.4. Statistical methodology

Each nominal system configuration was tested for $n = 30$ runs under controlled conditions. Trajectory efficiency was analysed using descriptive statistics, 95% confidence intervals, and effect-size reporting, together with a direct two-sample comparison between the FMI-based implementation and the Simulink baseline. Latency was treated as a distributional quantity and summarised using p50, p95, and p99 latency statistics across runs.

For the dropout sensitivity study, each perception-dropout condition was evaluated over 20 runs. The resulting degradation profile was analysed through changes in trajectory efficiency, success rate, and latency percentiles, and the degraded conditions were compared against the nominal perception condition using direct statistical tests where appropriate.

In all cases, the statistical interpretation reported in the results section avoids equivalence claims unless explicitly supported by an equivalence framework. Accordingly, differences are described either as statistically significant or not significant under the selected tests and confidence level.

5. Results and discussion

This section presents the quantitative validation of the proposed methodology. The analysis is structured in four stages. First, we verify the quality of the individual components as prerequisites for meaningful system-level evaluation. Second, we analyse the navigation behaviour of the fully integrated system. Third,

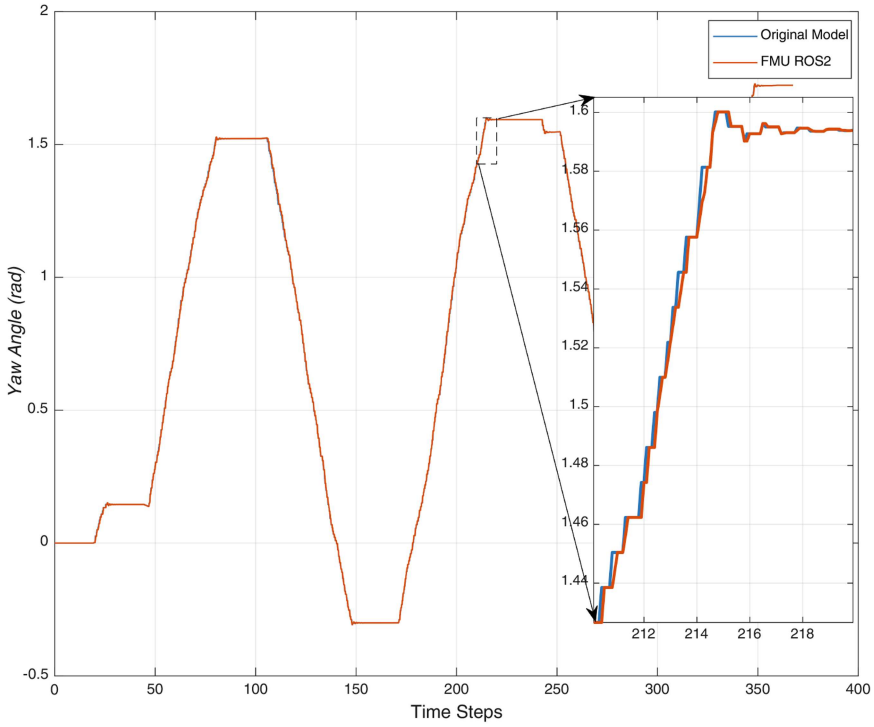


Figure 13. Odometry FMU validation (yaw angle θ), comparing the standalone artefact against its ROS 2-integrated counterpart. The inset highlights near-perfect agreement in a representative transition region.

we compare the proposed FMI-based implementation against the MathWorks Simulink baseline under controlled nominal conditions. Finally, we discuss the implications and limitations of the results from the perspective of model-based robotic integration and reproducibility.

5.1. Prerequisite: component-level validation

To ensure a meaningful system-level analysis, we first validated the performance of the core components in isolation and confirmed that their integration into ROS 2 preserves their intended behaviour.

5.1.1. Perception model performance

A reliable perception stream is a prerequisite for traffic-sign-guided navigation. We evaluated the fine-tuned YOLOv8 model on a held-out validation set. The model achieved a mean Average Precision ($mAP_{50:95}$) of 0.913, with high precision and recall across all traffic-sign classes. This result supports the suitability of the ONNX-based perception component as a source of visual features for the supervisory controller, thereby reducing the likelihood that downstream navigation behaviour is dominated by gross perception failure.

5.1.2. Integration integrity validation

We then verified that wrapping the models as ROS 2 nodes did not materially alter their numerical behaviour. By comparing the outputs of the standalone artefacts with those of their integrated ROS 2 node counterparts, we observed near-perfect agreement in both the odometry and perception pipelines.

Figure 13 shows the validation of the odometry FMU. The yaw angle θ produced by the ROS 2-integrated FMU closely follows the reference output of the standalone artefact over the full trajectory. The inset further shows that even in the zoomed transition region, both curves remain almost indistinguishable, indicating that the FMU wrapper preserves the numerical behaviour of the quaternion-to-yaw transformation with negligible deviation.

Figure 14 shows the corresponding validation of the perception node for the blobSize feature. The ROS 2-integrated ONNX pipeline reproduces the reference feature evolution with minimal deviation over the

complete sequence. The zoomed region confirms that the preprocessing, inference, decoding, and ROS 2 publication stages do not materially distort the extracted visual descriptor.

Taken together, these results indicate that the ROS 2 wrapping layer preserves the numerical integrity of both the FMI-based odometry artefact and the ONNX-based perception artefact, thereby supporting the validity of the subsequent system-level evaluation.

5.2. System-level navigation performance

With the component-level prerequisites satisfied, we evaluated the complete integrated system, focusing on the behaviour of the `SignTrackingLogic` FMU within the ROS 2/Gazebo execution environment. The system was executed for $n = 30$ simulation runs. Figure 15 shows a representative trajectory from two complementary perspectives. Figure 15(a) presents the path in the (x, y) plane, highlighting the main navigation phases, while Figure 15(b) overlays the same trajectory on the simulation environment. Together, these views provide qualitative evidence that the supervisory controller generates contextually appropriate manoeuvres in response to the traffic signs placed along the route.

Although these trajectory visualisations are useful for interpretation, they should be regarded as illustrative rather than sufficient evidence of system performance. For this reason, the nominal comparison against the baseline is based on quantitative metrics reported below.

5.3. Baseline comparison and statistical analysis

The central hypothesis of this work is that an open-source, standards-based toolchain can achieve competitive task-level performance with respect to a proprietary industrial workflow in the selected navigation scenario. To evaluate this claim, we compared the proposed FMI-based system against the MathWorks Simulink baseline under the same nominal experimental conditions.

In the nominal navigation task, both systems achieved a 100% task completion rate across all evaluated runs. Because task completion alone did not differentiate the two implementations, the comparison focuses

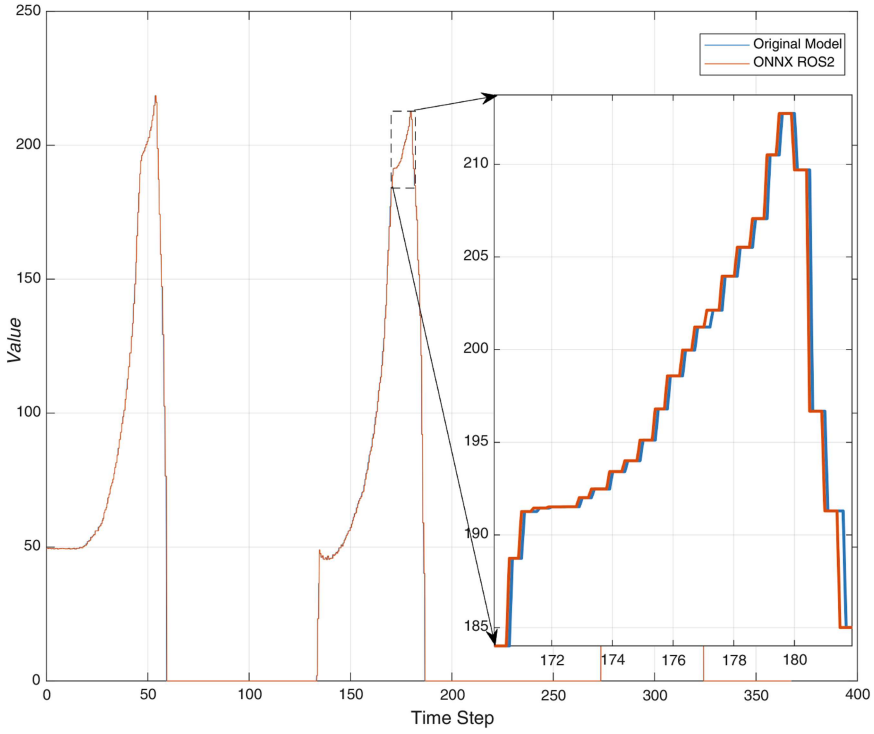
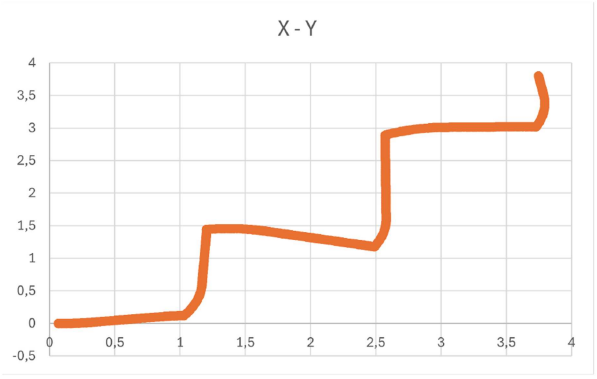
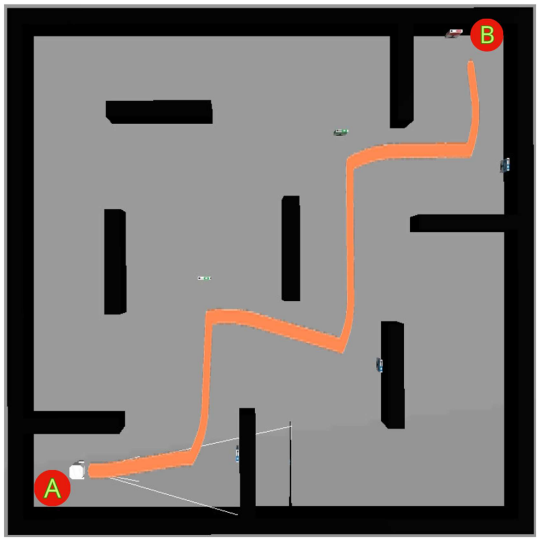


Figure 14. Perception-node validation for the blobSize feature, comparing the standalone artefact against its ROS 2-integrated counterpart. The inset shows that the local feature evolution is preserved with minimal deviation.



(a) X-Y position plot of the trajectory.



(b) Trajectory overlaid on the Gazebo map.

Figure 15. Two-part analysis of a representative robot trajectory, showing (a) the abstract path shape and (b) its contextual execution within the simulation environment.

Table 5. Nominal descriptive comparison between the proposed FMI-based system and the Simulink baseline. Trajectory efficiency is reported as mean ± standard deviation. Latency statistics are reported as percentile values across runs.

System	Trajectory Efficiency (η)	Latency p50 (ms)	Latency p95 (ms)	Latency p99 (ms)
MathWorks Baseline (Simulink)	0.835 ± 0.023	47.56	52.85	58.60
This Work (FMI + ONNX + ROS 2)	**0. 817 ± 0. 026**	**40.56**	**44.93**	**49.64**

on descriptive and inferential analysis of path efficiency and timing behaviour. Table 5 summarises the nominal descriptive statistics, while Table 6 reports the corresponding inferential comparison.

As shown in Table 5, the Simulink baseline obtained a higher mean trajectory efficiency than the FMI-based implementation (0.835 ± 0.023 vs 0.817 ± 0.026). The inferential analysis in Table 6 indicates that this difference was statistically significant ($p = 0.008$), with a mean difference of − 0.0176 (FMI − Simulink), a 95% confidence interval of $[-0.0305, -0.0048]$, and a moderate effect size (Hedges' $g = -0.70$). Therefore, under nominal conditions, the proprietary baseline achieved slightly better path-efficiency performance than the proposed open-source pipeline.

In contrast, the FMI-based implementation consistently outperformed the Simulink baseline in end-to-end latency. Across runs, the FMI-based workflow reduced latency by 7.00 ms at p50, 7.92 ms at p95, and 8.96 ms at p99. As summarised in Table 6, these reductions were statistically significant in all cases ($p < 0.001$), with confidence intervals that did not cross zero. This indicates that, although the Simulink baseline retained a modest advantage in trajectory efficiency, the FMI-based workflow provided a substantial timing advantage under nominal conditions.

Figure 16 visualises the distribution of nominal latency statistics across runs, highlighting the consistent reduction achieved by the FMI-based implementation in p50, p95, and p99 latency.

Taken together, the nominal results reveal a clear trade-off between the two workflows: the proprietary baseline favours path efficiency, whereas the FMI-based implementation favours execution responsiveness. From a methodological perspective, this is a meaningful outcome, because it shows that the proposed open-source pipeline can match the baseline in task completion while offering significantly lower latency under the evaluated execution conditions.

5.4. Sensitivity analysis under perception dropout

To evaluate the robustness of the proposed methodology under degraded perception conditions, we conducted a controlled sensitivity study in which the perception stream was subjected to increasing

Table 6. Inferential statistical comparison of nominal performance between the Simulink baseline and the FMI-based implementation. Differences are reported as FMI − Simulink.

Metric	Difference	95% CI	p-value	Effect Size
Trajectory Efficiency (η)	− 0.0176	[−0.0305, −0.0048]	0.008	Hedges' g = −0.70
Latency p50 (ms)	− 7.00	[−7.33, −6.68]	<0.001	—
Latency p95 (ms)	− 7.92	[−8.29, −7.55]	<0.001	—
Latency p99 (ms)	− 8.96	[−9.38, −8.54]	<0.001	—

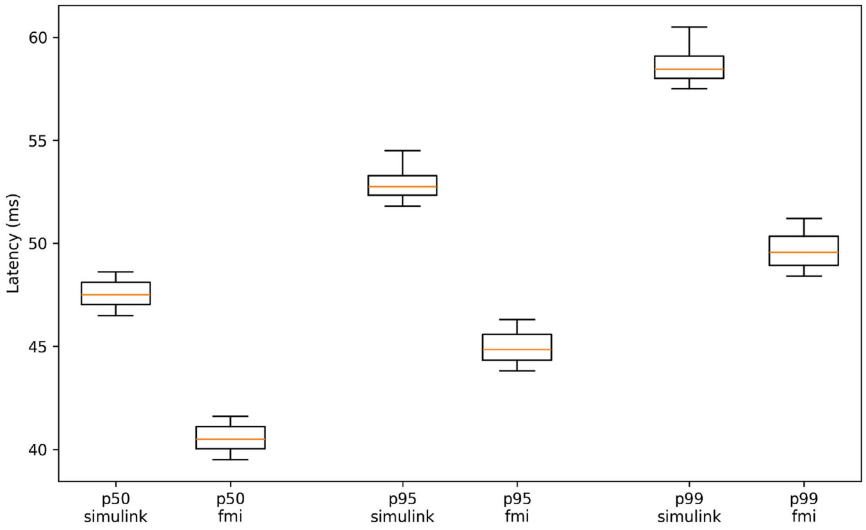


Figure 16. Distribution of nominal latency statistics across runs for the Simulink baseline and the FMI-based implementation. The FMI-based system shows consistently lower p50, p95, and p99 latency.

dropout rates of 0%, 10%, 20%, and 40%. For each condition, 20 runs were executed and evaluated in terms of task success rate, trajectory efficiency, and latency percentiles. Table 7 reports the descriptive results, whereas Table 8 summarises the corresponding statistical comparisons against the nominal 0% dropout condition.

The descriptive results in Table 7 show a clear degradation trend as perception dropout increases. Under nominal perception conditions (0% dropout), the system achieved a 100% success rate, with mean trajectory efficiency of 0.815 ± 0.043 and latency statistics of 39.89 ms (p50), 45.18 ms (p95), and 48.92 ms (p99). At 10% dropout, task completion remained intact, but trajectory efficiency decreased to 0.783 ± 0.046, while latency increased to 40.84 ms (p50), 47.01 ms (p95), and 52.46 ms (p99). At 20% dropout, degradation became more pronounced: success rate dropped to 90%, mean trajectory efficiency fell to 0.717 ± 0.106, and latency rose further to 41.13 ms (p50), 48.13 ms (p95), and 55.25 ms (p99). At 40% dropout, the system entered a markedly degraded regime, with success rate reduced to 35%, mean trajectory efficiency collapsing to 0.426 ± 0.174, and latency increasing to 42.12 ms (p50), 50.98 ms (p95), and 61.48 ms (p99).

The inferential results in Table 8 confirm that these degradations are not merely visual trends. Relative to the nominal 0% dropout condition, the reduction in trajectory efficiency was statistically significant for all nonzero dropout levels, and the latency increases were likewise significant for both p95 and p99. The progression of p-values and success-rate collapse indicates that perception dropout does not simply perturb the system smoothly; rather, it drives the controller through qualitatively different operating regimes.

Figure 17 provides the distributional view of trajectory efficiency across conditions. Its relevance is that it shows not only the decrease in central tendency already reported in Table 7, but also the growing variability of system behaviour as dropout increases. In particular, the widening spread at 20% and

Table 7. Descriptive sensitivity analysis under controlled perception dropout. Trajectory efficiency is reported as mean ± standard deviation. Latency values are reported as percentile values across runs.

Condition	Success Rate	Trajectory Efficiency (η)	Latency p50 (ms)	Latency p95 (ms)	Latency p99 (ms)
Dropout 0%	100%	0.815 ± 0.043	39.89	45.18	48.92
Dropout 10%	100%	0.783 ± 0.046	40.84	47.01	52.46
Dropout 20%	90%	0.717 ± 0.106	41.13	48.13	55.25
Dropout 40%	35%	0.426 ± 0.174	42.12	50.98	61.48

Table 8. Statistical comparison of degraded perception conditions against the nominal 0% dropout condition.

Condition vs. 0%	p-value (η)	p-value (p95)	p-value (p99)	Success Rate Change	Interpretation
10% dropout	0.028	0.0034	7.3×10^{-5}	100% → 100%	Graceful degradation
20% dropout	7.2×10^{-4}	1.14×10^{-5}	1.28×10^{-10}	100% → 90%	Functional degradation
40% dropout	2.8×10^{-9}	7.5×10^{-13}	7.65×10^{-21}	100% → 35%	Failure-prone regime

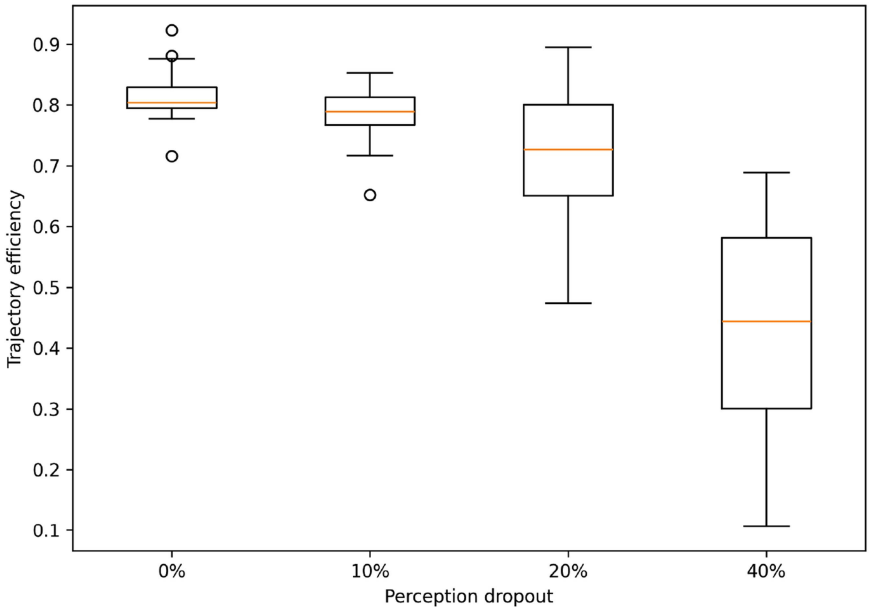


Figure 17. Distribution of trajectory efficiency across perception-dropout conditions. The downward shift in the distribution and the increase in spread indicate progressive degradation in both efficiency and consistency as perception availability decreases.

especially 40% dropout indicates that the system becomes less predictable, with some runs remaining partially effective while others deteriorate severely. This figure therefore supports the interpretation that perception loss affects both nominal performance and behavioural consistency.

Figure 18 complements this view from the timing perspective. The figure focuses on p95 latency because this percentile captures tail behaviour more clearly than central tendency alone. As dropout increases, the upward shift in the p95 distribution indicates that the controller becomes increasingly exposed to delayed or stale effective perception updates, thereby worsening response-time behaviour. In other words, this figure shows that degraded perception affects not only task success and path quality, but also the temporal responsiveness of the closed-loop system.

Figure 19 summarises the most interpretable system-level consequence of perception degradation: whether the mission is completed or not. While Figures 17 and 18 reveal gradual degradation in quality and timing, Figure 19 shows the threshold-like collapse in reliability. The flat performance from 0% to 10% dropout indicates tolerance to mild perception loss, whereas the drop at 20% and the severe collapse at 40% reveal the transition from degraded operation to failure-prone behaviour.

Taken together, the sensitivity analysis shows that the proposed open-source pipeline remains operational under mild perception loss, but degrades progressively as dropout increases. The 10% dropout

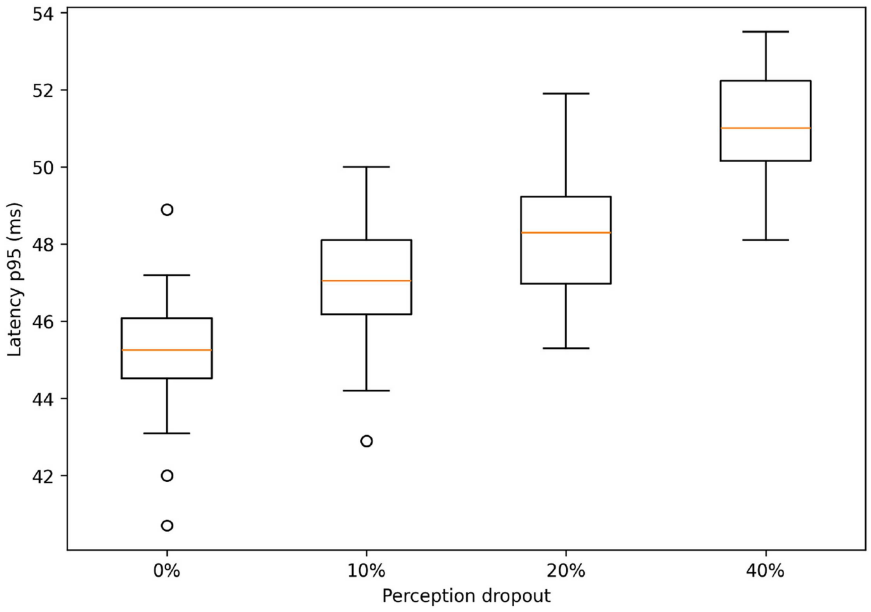


Figure 18. Distribution of latency p95 across perception-dropout conditions. The upward shift of the distribution indicates progressively worse latency tails as perception dropout increases.

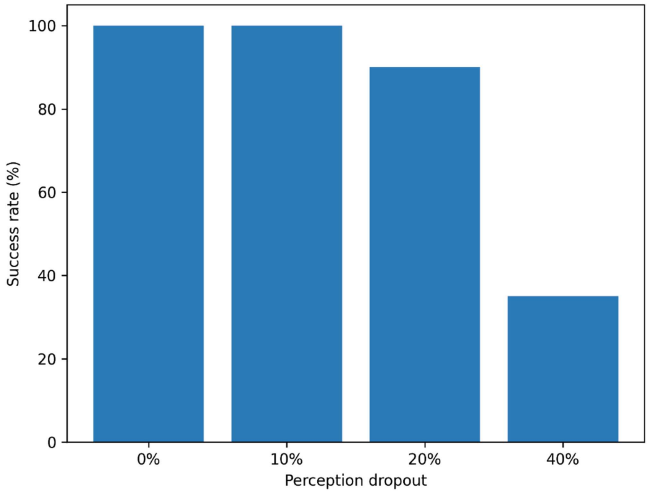


Figure 19. Task success rate under increasing perception dropout. The system preserves full success at 10% dropout, begins to fail at 20%, and degrades severely at 40%.

condition corresponds to graceful degradation, in which task completion remains intact despite reduced efficiency and slightly increased latency. The 20% condition marks the onset of functional degradation, as the system still completes most runs successfully, but does so with reduced efficiency, increased latency, and a non-negligible failure rate. At 40% dropout, the system enters a failure-prone regime in which perception loss severely disrupts task completion and latency behaviour. These findings confirm that perception availability is a central determinant of system-level performance and that robustness analysis is essential when evaluating AI-in-the-loop robotic workflows.

5.5. Discussion

The results support the feasibility of the proposed methodology as an open-source workflow for AI-driven robotic integration. More specifically, they show that FMU-encapsulated model-based components and

ONNX-based perception modules can be orchestrated within ROS 2 to produce coherent task-level behaviour in a realistic robotic simulation scenario.

The nominal comparison reveals an important trade-off. While the proprietary Simulink baseline achieved slightly higher trajectory efficiency, the FMI-based workflow consistently exhibited substantially lower latency across p50, p95, and p99. This indicates that the proposed open-source methodology does not simply reproduce the proprietary behaviour, but instead occupies a different operating point in the design space: it incurs a modest reduction in path efficiency while providing a substantial timing advantage.

This trade-off should be interpreted carefully. The trajectory-efficiency difference was statistically significant under the nominal protocol, with the Simulink baseline outperforming the FMI-based implementation by a moderate margin. At the same time, the latency reduction observed for the FMI-based pipeline was large and consistent across all reported latency percentiles. Accordingly, the contribution of the proposed methodology is not that it dominates the proprietary baseline in every metric, but that it achieves full task completion under nominal conditions while offering markedly improved execution responsiveness in an open, portable, and reproducible workflow.

The sensitivity analysis further strengthens the methodological contribution of the paper. It shows that the proposed architecture is not only functional under nominal conditions, but also exhibits a measurable degradation profile under controlled perception loss. In particular, the transition from full success at 10% dropout to partial failure at 20% and severe collapse at 40% provides concrete evidence that perception availability and timing heterogeneity directly shape system-level robotic behaviour. This confirms that execution semantics and robustness analysis are integral parts of the methodology rather than secondary implementation details.

From a systems perspective, the results highlight why explicit execution semantics matter. The interaction between asynchronous perception outputs, FMU-based control evaluation, and ROS 2 communication affects not only software composition but also observable robotic behaviour. This reinforces the point that interoperability cannot be reduced to connector availability alone; timing assumptions, input-handling rules, and wrapper-level orchestration directly influence task-level performance and must therefore be reported as first-class elements of the contribution.

The most relevant methodological outcome is not merely that the task was completed successfully, but that the resulting system could be assembled from portable executable artefacts whose interaction was made explicit through model-based control formalisation, ONNX-based perception abstraction, and middleware-level orchestration. In this sense, the contribution of the work lies as much in the definition of a reproducible integration method as in the absolute task performance achieved in the specific navigation scenario.

This work therefore serves as a focused case study for the relevance of Modelica and FMI beyond traditional simulation domains. It demonstrates that these standards can be used to formalise, encapsulate, and deploy control-oriented functionality that interacts with learned perception components in an open robotic workflow. At the same time, the present study has clear limits: the task is intentionally constrained, the baseline comparison is centred on a specific nominal scenario, and the physical-robot deployment is presently used as evidence of practical portability rather than as a full quantitative simulation-to-hardware validation.

Overall, the strongest conclusion supported by the current evidence is that an FMI-centric open-source methodology is viable, transparent, and technically competitive for AI-driven robotics, even if it does not uniformly outperform the proprietary baseline in every metric. The current results show that such a methodology can combine full nominal task success, significantly lower latency, and explicitly characterised degradation under controlled perception loss, thereby providing a stronger foundation for future extensions toward richer control logic, broader robustness testing, and quantitative physical-robot evaluation.

5.6. Physical deployment as proof of portability

The integrated workflow was also deployed on a physical TurtleBot3 platform in order to verify whether the same FMI-based control artefacts, ONNX-based perception component, and ROS 2 integration structure could be reused beyond simulation. Figure 20 shows the resulting physical implementation.

The relevance of this result is not to claim a full quantitative simulation-to-hardware validation campaign, but to provide qualitative evidence of practical portability. More specifically, it shows that the proposed methodology is not restricted to the Gazebo environment: the same general artefact-level organisation, including FMU-wrapped model-based components, ONNX-executed perception, and ROS

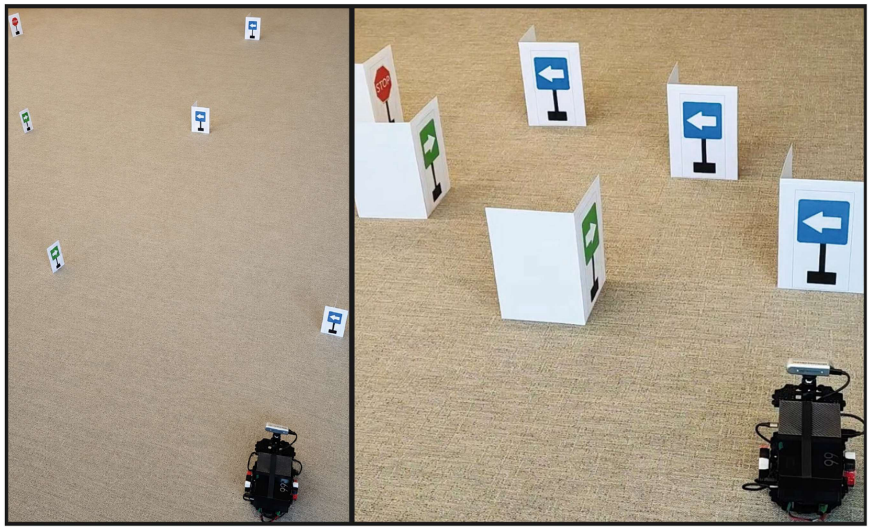


Figure 20. Physical deployment of the integrated system on a TurtleBot3 platform, illustrating practical portability of the FMI-based workflow beyond simulation.

2-based orchestration, can be carried over to a real robotic platform without redesigning the full software architecture.

This observation is consistent with one of the central methodological claims of the paper, namely that open standards and reusable executable artefacts can reduce reimplementation effort across development stages. At the same time, this physical deployment should be interpreted cautiously. In the present manuscript, it serves as a proof of implementation feasibility and portability rather than as a controlled quantitative comparison against simulation or against the proprietary baseline. A more rigorous physical-robot evaluation, including repeatability analysis, latency characterisation, and quantitative comparison between simulation and hardware, remains part of future work.

6. Conclusions and future work

6.1. Conclusions

This paper presented an open-source, FMI-centric methodology for developing AI-driven robotic systems by integrating OpenModelica-generated FMUs, ONNX-based perception, and ROS 2 middleware within a common execution environment. Beyond software connectivity, the proposed methodology explicitly formalised the supervisory control layer, defined the role of reusable executable artefacts, and clarified the interaction between model-based and perception-based components in a heterogeneous robotic workflow. The methodology was instantiated and evaluated on a traffic-sign-guided navigation task, using the MathWorks Simulink example as a proprietary baseline for nominal comparison.

The results support the viability of the proposed workflow in the evaluated scenario. Under nominal operating conditions, both systems achieved a 100% task completion rate. The Simulink baseline retained a moderate advantage in trajectory efficiency, whereas the FMI-based implementation exhibited significantly lower latency across the reported percentile metrics. These results indicate that the proposed open-source workflow does not simply replicate the proprietary baseline, but instead achieves a different and practically relevant trade-off between task-level path efficiency and execution responsiveness.

In addition, the controlled perception-dropout study showed that the system remains fully operational under mild perception loss, but degrades progressively as dropout increases, with clear reductions in efficiency, increased latency tails, and eventual failure-prone behaviour at high dropout rates. This strengthens the paper by showing not only nominal viability, but also a quantified robustness envelope for the proposed methodology under degraded perception availability.

From a methodological perspective, the main contribution of this work is the demonstration that FMU-encapsulated control logic and ONNX-deployed perception can be composed within ROS 2 as portable

and reusable artefacts while preserving coherent system-level behaviour. In this sense, the study provides evidence that Modelica and FMI can contribute beyond traditional desktop simulation by supporting executable control-oriented components within open robotic architectures.

The physical deployment of the integrated system on a TurtleBot3 platform further supports the practical portability of the proposed workflow beyond simulation. However, this result should be interpreted as evidence of implementation feasibility rather than as a full quantitative simulation-to-hardware validation, since the present study focuses primarily on simulated nominal comparison and controlled degradation analysis.

Overall, the strongest conclusion supported by the present evidence is that an open, standards-based methodology centred on FMI, ONNX, and ROS 2 constitutes a viable, transparent, and technically competitive alternative to proprietary workflows for this class of robotic task, while offering clear benefits in modularity, portability, and reproducibility.

6.2. Future work

While the present implementation demonstrates the viability of the methodology using FMI 2.0, the future adoption of FMI 3.0 could unlock capabilities that are particularly relevant for more demanding robotic applications. Several of its features directly address limitations associated with heterogeneous timing and real-time coordination:

- **Clocks and Hybrid Co-Simulation:** The introduction of clocks could support a more explicit synchronisation framework between asynchronous perception updates and periodic control loops, thereby reducing ambiguity in cross-domain timing semantics.
- **Scheduled Execution Interface:** This interface is especially promising for robotics because it would allow external schedulers to manage FMU execution in a more deterministic way, which is important for future hard real-time or safety-critical deployments.

Accordingly, one major direction for future work is to review the present methodology under FMI 3.0 once robust open-source export and deployment support becomes mature, especially in OpenModelica-based toolchains. Additional research directions include extended sensitivity studies under varying control-step sizes, inference rates, and induced message delays; richer quantitative evaluation on physical robots; and the integration of more advanced AI modules for perception and planning. Finally, it would also be valuable to explore alternative middleware technologies, such as Zenoh, in order to assess their impact on latency and scalability in distributed deployments.

Author contributions

CRediT: **Sebastian Rojas-Ordoñez:** Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Resources, Software, Validation, Visualization, Writing – original draft, Writing – review & editing; **Mikel Segura:** Investigation, Methodology, Project administration, Supervision, Writing – review & editing; **Jorge Arboleda:** Investigation, Resources; **Ekaitz Zulueta:** Investigation, Project administration, Supervision, Writing – review & editing.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

This work was supported by the SENDOA project ID: KK-2025/00102; the DBaskIN KK-2025/00012 project, financed by the Council of the Basque Country.

Data availability statement

The methodological workflow, system architecture, execution assumptions, and experimental protocol are described in detail in the manuscript. The numerical data underlying the reported results and additional implementation details are available from the corresponding author upon reasonable request.

References

Amorim T. 2019. Strategies and best practices for model-based systems engineering adoption in embedded systems industry. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). IEEE.

Betz T, Schmeller M, Teper H, Betz J. 2023. How fast is my software? Latency evaluation for a ROS 2 autonomous driving software. In: In 2023 IEEE Intelligent Vehicles Symposium (IV). p 1–6. https://doi.org/10.1109/IV55152.2023.10186585

Blockwitz T et al. 2012. Functional mockup interface 2.0: the standard for tool independent exchange of simulation models. In: Proceedings of the 9th International Modelica Conference. Munich, Germany. p 173–184. https://doi.org/10.3384/ecp12076173

Corallo A, Del Vecchio V, Luperto A, Marra M. 2022. Evaluating the maturity of MBE application. In: Canciglieri IO, Jr., Noël F, Rivest L, Bouras A, editors. Product Lifecycle Management: Green and Blue Technologies to Support Smart and Sustainable Organizations. Springer: Cham. p 401–415.

El-Hussieny H. 2024. Real-time deep learning-based model predictive control of a 3-DOF biped robot leg. NatSR. 14(1):16243. https://doi.org/10.1038/s41598-024-66104-y

Gomes C, Lucio L, Vangheluwe H. 2019. Semantics of co-simulation algorithms with simulator contracts. In: Proceedings of the 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C). p 784–789. https://doi.org/10.1109/MODELS-C.2019.00124

Haessler P, Giones F, Brem A. 2023. The who and how of commercializing emerging technologies: a technology-focused review. Technovation. 121:102637. https://doi.org/10.1016/j.technovation.2022.102637

Hansen ST et al. 2022. The FMI 3.0 standard interface for clocked and scheduled simulations. Electronics. 11(21):3635. https://doi.org/10.3390/electronics11213635

Hatledal LI. 2025. Ecos: an accessible and intuitive co-simulation framework. Journal of Open Source Software. 10(110):8182. https://doi.org/10.21105/joss.08182

Holtmann J, Liebel G, Steghöfer J-P. 2024. Processes, methods, and tools in model-based engineering—A qualitative multiple-case study. J Syst Softw. 210:111943. https://doi.org/10.1016/j.jss.2023.111943

Javaid M, Haleem A, Singh RP, Suman R. 2023. An integrated outlook of cyber–physical systems for industry 4.0: topical practices, architecture, and applications. Green Technologies and Sustainability. 1(1):100001. https://doi.org/10.1016/j.grets.2022.100001

Khandoker A et al. 2022. Towards a logical framework for ideal MBSE tool selection based on discipline specific requirements. J Syst Softw. 189:111306. https://doi.org/10.1016/j.jss.2022.111306

Kortenkamp D, Simmons R, Brugali D. 2016. Robotic systems architectures and programming. In: Siciliano IB, Khatib O, editors. Springer Handbook of Robotics. 2nd ed. Springer: Cham. p 283–306. https://doi.org/10.1007/978-3-319-32552-1_11

Lange R, Traversaro S, Lenord O, Bertsch C. 2021. Integrating the functional mock-up interface with ROS and gazebo. In: Koubaa IA, editor. Robot Operating System (ROS): The Complete Reference. Vol. 5. Cham: Springer; p. 187–231.

Licardo JT, Domjan M, Orehovački T. 2024. Intelligent robotics—A systematic review of emerging technologies and trends. Electronics. 13(3):542. https://doi.org/10.3390/electronics13030542

MathWorks. 2024. Sign-Following Robot with ROS 2 in Simulink. Available at: (accessed March 16, 2026) https://es.mathworks.com/help/ros/ug/sign-following-robot-using-ros2-simulink.html

Modelica Association. 2022. unctional Mock-up Interface Standard, Version 3.0. Available at: (accessed March 16, 2026) https://fmi-standard.org/docs/3.0/

Modelica Association. 2024. Functional Mock-up Interface Specification, Version 3.0.2. Available at: (accessed March 16, 2026) https://fmi-standard.org/docs/3.0.2/

Moshood TD, Rotimi JOB, Shahzad W, Bamgbade JA. 2024. Infrastructure digital twin technology: a new paradigm for future construction industry. Technol Soc. 77:102519. https://doi.org/10.1016/j.techsoc.2024.102519

ONNX Community. 2025. Open Neural Network Exchange (ONNX) Documentation. Available at: (accessed March 16, 2026) https://onnx.ai/

Open Robotics. 2025. ROS 2 Humble Documentation. Available at: (accessed March 16, 2026) https://docs.ros.org/en/humble/

Open Source Modelica Consortium. 2024. OpenModelica User's Guide: Functional Mock-up Interface (FMI) Support. Available at: (accessed March 16, 2026) https://openmodelica.org/doc/OpenModelicaUsersGuide/latest/

Oudart D, Cantenot J, Boulanger F, Chabridon S. 2020. A model-based toolchain for the co-simulation of cyber-physical systems with FMI. In: Proceedings of the 8th International Conference on Model-Driven Engineering and Software Development. Valletta, Malta. p 15–25. https://doi.org/10.5220/0008875400150025

Patel V, Liarokapis M, Dollar A. 2022. Open robot hardware: progress, benefits, challenges, and best practices. IEEE Rob Autom Mag. 29(3):17–29.

Patrício L, Varela L, Silveira Z. 2024. Integration of artificial intelligence and robotic process automation: literature review and proposal for a sustainable model. Applied Sciences. 14(21):9648.

Robles J, Martín C, Díaz M. 2023. OpenTwins: an open-source framework for the development of next-gen compositional digital twins. Comput Ind. 152:104007. https://doi.org/10.1016/j.compind.2023.104007

Rojas-Ordoñez S, Segura M, Zulueta E. 2025. Integration of physical and AI models using open and interoperable standards: a model-based methodology for autonomous robot development. In: Proceedings of the 16th International Modelica & FMI Conference. p 479–488.

Santos A et al. 2024. Integration of artificial vision and image processing into a pick and place collaborative robotic system. Journal of Intelligent & Robotic Systems. 110:1–18. https://doi.org/10.1007/s10846-024-02195-z

Sekar PKM, Baras JS. 2022. Model-based systems engineering simulation framework for robot grasping. INCOSE International Symposium. 32(S2):82–89. https://doi.org/10.1002/iis2.12898

Shahsavari S, Immonen E, Rabah M, Haghbayan M-H, Plosila J. 2021. MCX: an open-source framework for digital twins. In: Proceedings of the 35th European Conference on Modelling and Simulation (ECMS). p 119–124.

Siegwart R, Nourbakhsh IR, Scaramuzza D. 2011. Introduction to Autonomous Mobile Robots. 2nd ed. The MIT Press: Cambridge, MA, USA.

Yue L. 2024. Design of real-time dynamic motion compensation in vision-guided robots for biopsy needle alignment. MSc assignment/presentation, Robotics and Mechatronics Group, University of Twente: Enschede, The Netherlands. https://api.semanticscholar.org/CorpusID:278238181