

# A Decision Framework to Select Robotics Simulators for Automation and Control Tasks: Criteria and Case-Study Application <sup>†</sup>

Tiago A. T. B. Baptista <sup>1</sup> , César M. A. Vasques <sup>1,2,\*</sup> , Pedro M. R. Castro <sup>3</sup>  and Adélio M. S. Cavadas <sup>1</sup> 

- <sup>1</sup> Research Unit on Materials, Energy and Environment for Sustainability (proMetheus), Higher School of Technology and Management, Polytechnic Institute of Viana do Castelo, 4900-347 Viana do Castelo, Portugal; tiagobaptista@ipvc.pt (T.A.T.B.B.); adelioc@estg.ipvc.pt (A.M.S.C.)
- <sup>2</sup> Centre for Mechanical Technology and Automation (TEMA), Department of Mechanical Engineering, University of Aveiro, Campus Universitário de Santiago, 3810-193 Aveiro, Portugal
- <sup>3</sup> Applied Digital Transformation Laboratory (ADiT-Lab), Higher School of Technology and Management, Polytechnic Institute of Viana do Castelo, 4900-347 Viana do Castelo, Portugal; pmc@estg.ipvc.pt
- \* Correspondence: cmavasques@ua.pt
- <sup>†</sup> Presented at the 3rd International Electronic Conference on Machines and Applications, 12–14 May 2026; Available online: <https://sciforum.net/event/IECMA2026>.

## Abstract

Robotics simulation is a key enabler for automation and control development, allowing safer experimentation, faster design iterations, and reduced development cost. Nevertheless, the current simulator ecosystem is highly fragmented, spanning open-source and commercial tools with different levels of physical fidelity, performance, and ecosystem integration. As a result, simulator selection is frequently driven by familiarity or availability rather than by explicit task requirements, often leading to suboptimal engineering workflows. This paper proposes a task-oriented decision framework to support reproducible and transparent selection of robotics simulators based on a fixed and structured set of evaluation criteria. These criteria cover (i) physical fidelity and contact modelling; (ii) sensor modelling and visual realism; (iii) performance and scalability aspects, including headless execution, parallelism, and GPU acceleration; (iv) ecosystem integration with automation, control, and learning pipelines, including ROS/ROS 2 compatibility; (v) extensibility and programmability; and (vi) practical constraints such as hardware requirements, licensing models, and learning curve. The framework is operationalised through a checklist and scoring matrix guided by four key questions addressing the target task, fidelity-versus-speed priorities, target software stack, and sim-to-real transfer requirements. To examine feasibility in a representative engineering workflow, a URDF-based modelling and simulation pipeline is implemented and used to compare Gazebo, as an open-source physics-based simulator, against MATLAB/Simulink, representing a commercial model-based simulation environment. The comparison reports practical indicators including setup effort, integration complexity, computational requirements, and runtime behaviour for repeated executions of a representative motion-oriented sequence. The results highlight consistent trade-offs across different user profiles and application needs while also revealing open gaps in the field, notably the lack of unified multi-task benchmarks and joint metrics capable of simultaneously capturing simulation fidelity, computational performance, and sim-to-real transfer effectiveness.

**Keywords:** robotics; simulation platforms; simulator selection; decision framework; MATLAB/Simulink; Gazebo; automation and control; URDF; digital twins; sim-to-real transfer



Academic Editor: Antonio J. Marques Cardoso

Published: 30 July 2026

**Copyright:** © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

## 1. Introduction

Robotics simulation is a key tool for automation and control development because it supports safer experimentation, faster design iterations, and repeatable virtual prototyping [1,2]. Testing directly on physical hardware can increase cost, create safety risks, and make early design iterations difficult to reproduce. For this reason, simulators are widely used in robotics research and engineering practice for controller development, sensing and perception testing, and preliminary verification before deployment.

Selecting an appropriate simulator is nevertheless a non-trivial engineering decision. The current ecosystem includes open-source and commercial tools with different physics engines, contact models, sensor capabilities, rendering pipelines, performance profiles, middleware interfaces, licensing conditions, and hardware requirements [2,3]. In practice, simulator choice is often driven by familiarity, availability, or institutional constraints rather than by explicit task requirements. This can lead to avoidable rework when the selected tool does not match the intended workflow.

This issue is particularly relevant in mechanical engineering, automation, and applied robotics contexts, where simulator suitability depends on the task being addressed. A Robot Operating System (ROS)-oriented control workflow, a model-based design workflow, a reinforcement-learning experiment, a digital-twin application, and an industrial offline programming scenario do not impose the same priorities. ROS-centric platforms are often appropriate for classical robotics workflows; graphics processing unit (GPU)-accelerated environments are relevant for large-scale learning and photorealistic digital-twin pipelines; and contact-intensive tasks require careful attention to contact modelling, numerical stability, and physical fidelity [2–4]. Reproducibility requirements, computational budgets, licensing restrictions, and integration needs further influence the final decision [1,5].

Existing reviews provide valuable information about simulator capabilities, including contact modelling, sensor simulation, performance, ROS compatibility, physics engines, and sim-to-real transfer challenges [2–4,6,7]. However, this information is commonly organised by simulator family, technical feature, or application domain. It does not always provide a compact and traceable procedure that helps an engineer translate task requirements into simulator-selection criteria and compare alternatives consistently.

The contribution of this work is a task-oriented decision framework for selecting robotics simulators in automation and control workflows. The framework organises simulator assessment around six criterion groups: physical fidelity and contact modelling, sensor modelling and visual realism, performance and scalability, ecosystem integration, extensibility and programmability, and practical constraints. These criteria are operationalised through a checklist, a conservative 0–2 scoring matrix, four guiding questions, and a weighted aggregation procedure. The goal is not to define a universal best simulator but to support transparent recommendations that change according to the target task, software stack, fidelity-versus-speed priority, and sim-to-real requirements.

The framework is applied to a global analysis of representative robotics simulation platforms and illustrated through a URDF-based case study comparing Gazebo/ROS 2 and MATLAB/Simulink in an automation and control context. The case study reports practical workflow observations and instrumented runtime measurements, including observed execution interval, CPU utilisation, RAM usage, GPU utilisation, and VRAM allocation, while avoiding general benchmark claims beyond the tested scenario.

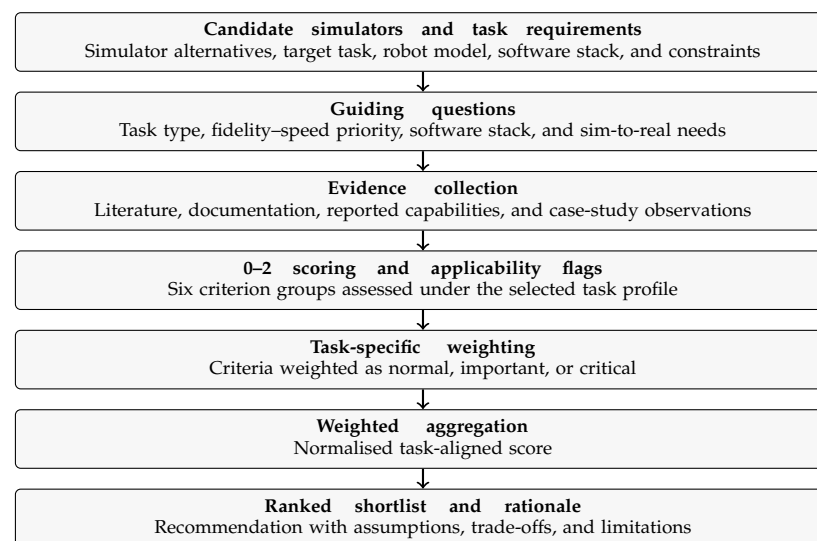
The paper is organised as follows: Section 2 presents the proposed decision framework, evaluation criteria, scoring logic, and prioritisation workflow; Section 3 applies the framework to a global simulator analysis; Section 4 presents the illustrative URDF-based Gazebo and MATLAB/Simulink case study; Section 5 discusses implications and limitations; and Section 6 concludes the paper.

## 2. Proposed Decision Framework

### 2.1. Method Overview

This work proposes a task-oriented decision framework for selecting robotics simulators in automation and control workflows, with an emphasis on transparency and reproducibility. The framework is grounded in prior literature on robotics simulators, physics engines, and selection criteria, while remaining compact enough for practical engineering use [1–4].

The method starts from two inputs: a set of candidate simulators and a set of task requirements. These requirements are captured through four guiding questions covering the target task, fidelity-versus-speed priorities, target software stack, and sim-to-real transfer needs. Figure 1 summarises the resulting decision workflow.



**Figure 1.** Structured workflow of the proposed task-oriented simulator-selection framework, linking task requirements, evidence collection, criterion scoring, task-specific weighting, weighted aggregation, and final recommendation.

Simulator capabilities are rated across six criterion groups using an ordinal 0–2 scale. A score of 0 indicates that the capability is absent, undocumented, or impractical for the target task; a score of 1 indicates partial or workaround-dependent support; and a score of 2 indicates mature and practical support for the intended workflow. The compact scale avoids implying false numerical precision while keeping the decision process reproducible.

Scores are then combined with a prioritisation profile derived from the four guiding questions. The priority levels normal, important, and critical define the weights used to compute task-aligned rankings. The following subsections describe the evaluation criteria, scoring structure, prioritisation workflow, and aggregation rule.

### 2.2. Evaluation Criteria

The framework evaluates candidate simulators using six fixed criterion groups derived from the literature and operationalised for consistent comparison across tools [1–4]. Each group is assessed using the 0–2 ordinal scale defined in Section 2.1. Table 1 defines the scoring rubric used to assign the 0–2 score for each criterion group. The rubric is intended to make the qualitative assessment more explicit and reproducible across candidate simulators.

**Table 1.** Scoring rubric used to assign 0–2 scores to each criterion group.

| Criterion Group                         | Score 0                                                                                                                                                                                  | Score 1                                                                                                                                                                            | Score 2                                                                                                                                                                                                         |
|-----------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Physical fidelity and contact modelling | Contact, friction, or joint/constraint behaviour is absent, poorly documented, or impractical for the target task.                                                                       | Basic contact and joint behaviour is available, but with relevant limitations, manual tuning, or reduced repeatability.                                                            | Mature and configurable contact, friction, and joint/constraint behaviour is supported with stable and repeatable execution for the target task.                                                                |
| Sensor modelling and visual realism     | Sensor simulation is absent, very limited, or not clearly supported for the intended workflow.                                                                                           | Common sensor modalities or rendering features are available, but with limited realism, limited configuration, or partial support for noise, latency, or synchronisation.          | Sensor and visual simulation is well supported and configurable, including relevant modalities, parameters, noise/latency modelling, synchronisation, and ground-truth outputs where applicable.                |
| Performance and scalability             | Execution is impractical for the target workload due to limited control, lack of headless operation, or poor scalability.                                                                | The workload is executable, but scalability is limited, headless or parallel execution requires workarounds, or timing/profiling support is partial.                               | The tool supports efficient execution, headless operation, timing control, and scalable runs, with hardware acceleration and profiling support where applicable.                                                |
| Ecosystem integration                   | Integration with the target robotics or automation stack is absent, obsolete, or impractical.                                                                                            | Integration is feasible but requires adapters, bridges, or manual configuration; support for ROS/ROS 2, URDF/SDF, controllers, or logging is partial.                              | Integration is mature and well documented for the intended workflow, including maintained support for relevant middleware, model formats, controller interfaces, logging, and automation or learning pipelines. |
| Extensibility and programmability       | Programmability is limited, closed, or poorly documented for extending scenarios or automating experiments.                                                                              | Usable APIs, scripting, plugins, or examples are available, but with limitations in documentation, coverage, or interface stability.                                               | Mature and documented APIs/SDKs, plugin or extension mechanisms, scripting support, and stable interfaces are available for scenario design and experiment automation.                                          |
| Practical constraints                   | The tool presents major barriers for the target context, such as restrictive licensing, high cost, excessive hardware requirements, difficult installation, or weak maintenance signals. | The tool is usable but presents relevant practical constraints, such as non-trivial setup, licensing limitations, hardware demands, portability issues, or a steep learning curve. | The tool is practical for the target context, with acceptable cost/licensing, manageable hardware requirements, documented installation, reasonable portability, and active maintenance/community support.      |

### 2.3. Checklist and Scoring Matrix

The framework is operationalised through a checklist and scoring matrix. The checklist records the evidence used to assess each simulator, while the matrix converts that assessment into a compact 0–2 score for each criterion group. The full matrix is maintained in structured tabular form, and the manuscript reports compact summary tables.

Each simulator is identified by name and version whenever this information is available. Scores are assigned under a standard baseline setup using the default configuration unless otherwise stated. This baseline improves comparability across platforms, although it does not cover every possible simulator configuration, plugin, hardware setup, or tuning strategy.

Score assignment follows three steps. First, the available evidence is collected from the literature, official documentation, reported capabilities, and case-study observations where applicable. Second, each criterion group is assessed using the rubric in Table 1. Third, the resulting score is checked against the intended task profile to confirm whether the capability is relevant and sufficiently supported for that use case.

The matrix records two types of information: a 0–2 capability score per criterion group and an applicability flag per task profile. Criteria marked as applicable are included in aggregation. Criteria marked as N/A are considered irrelevant for that task profile and are excluded from aggregation.

Capabilities that cannot be reliably confirmed are marked as unknown. To avoid overestimating simulator support, unknown capabilities are treated conservatively as 0 until verified. This rule is especially important when documentation is incomplete, support is version-dependent, or a capability exists only through experimental extensions.

The completed matrix is combined with the prioritisation profile derived from the four guiding questions. The priority levels normal, important, and critical define the weights used to compute aggregated scores, task-specific rankings, and recommendations.

#### 2.4. Guiding Questions and Prioritisation Workflow

Simulator selection is guided by four questions that capture the main task requirements and determine which criterion groups should be emphasised. Considered jointly, these questions produce a task profile that assigns priority levels (normal, important, critical) to the six criterion groups.

**Q1: What is the target task type?** The task type defines the baseline relevance of the criterion groups. Control-oriented workflows tend to emphasise integration and timing behaviour, learning-oriented workflows emphasise scalability and computational efficiency, and perception-driven workflows place greater emphasis on sensor modelling and visual realism.

**Q2: What is the fidelity-versus-speed priority?** Workflows prioritising realism increase the importance of physical fidelity, contact modelling, and sensor realism, whereas workflows prioritising throughput place greater emphasis on performance and scalability.

**Q3: What is the target software stack?** Required middleware, model formats, controller interfaces, and interoperability needs (e.g., ROS/ROS 2 integration) mainly affect ecosystem integration and extensibility/programmability.

**Q4: What are the sim-to-real transfer requirements?** When sim-to-real transfer is important, the framework increases emphasis on physical fidelity, sensor modelling, timing behaviour, and alignment with the intended deployment stack.

The answers are translated into a prioritisation profile that assigns normal, important, or critical levels to each criterion group and flags criteria as applicable or N/A for the task under consideration. This profile is then used in the aggregation step described in Section 2.5.

#### 2.5. Score Aggregation

Aggregation combines the 0–2 criterion scores from the scoring matrix with the prioritisation profile derived from the guiding questions. Priority levels are mapped to weights  $w_i \in \{1, 2, 3\}$  for each applicable criterion group  $i$  (normal = 1, important = 2, critical = 3). For a given simulator and task profile, let  $s_i \in \{0, 1, 2\}$  denote the score assigned to criterion group  $i$  and  $w_i$  the corresponding priority weight. The aggregated score is computed as

$$\text{score} = \frac{\sum_i w_i s_i}{2 \sum_i w_i}, \quad (1)$$

where the factor 2 in the denominator corresponds to the maximum attainable score for each criterion group, since each capability score is assigned on a 0–2 scale. This normalisation therefore divides the weighted sum by the maximum possible weighted score for the selected task profile, ensuring that the final score remains in the interval  $[0, 1]$ .

**Special cases.** Criteria marked as N/A are excluded from the aggregation. Capabilities marked as unknown are treated conservatively as  $s_i = 0$  until verified.

**Recommendation rule.** Simulators are ranked by their final score. Ties are first resolved by comparing performance on critical criteria and, if necessary, by practical constraints such as licensing and hardware requirements.

### 3. Global Simulator Analysis

#### 3.1. Candidate Simulators and Analysis Scope

The global analysis covers nine tools: Gazebo, MATLAB/Simulink (Simscape Multibody), NVIDIA Isaac Sim, CoppeliaSim, Webots, MuJoCo, PyBullet, RoboDK, and ABB RobotStudio. Together, they span open-source, commercial, learning-oriented, model-based, and industrial planning environments.

The software-version baseline for the global analysis was Gazebo Harmonic, MATLAB/Simulink R2025b, NVIDIA Isaac Sim 6.0.0, CoppeliaSim V4.10.0, Webots R2025a, MuJoCo 3.10.0, PyBullet 3.2.7, RoboDK 6.0.5, and ABB RobotStudio 2026.2. Gazebo Harmonic and MATLAB/Simulink R2025b were used in the empirical case study; the remaining platforms were assessed through literature and official documentation using the latest stable releases available on 9 July 2026. The externally sourced screenshots in Figure 2 are illustrative, with their provenance and identifiable version details reported in the caption.

The selection reflects the workflow families introduced in Section 2.4 and is consistent with prior reviews [2–4,8]. ROS/URDF-centric tools are represented by Gazebo and Webots [3,8–11]; learning and optimisation workflows by NVIDIA Isaac Sim, MuJoCo, and PyBullet [4,12–14]; general-purpose robot simulation is represented by CoppeliaSim [8,15]; industrial offline programming and cell planning by RoboDK and ABB RobotStudio [16,17]; and model-based simulation by MATLAB/Simulink with Simscape Multibody [18,19].

The analysis targets automation and control tasks and is scoped to the six criterion groups in Table 1. It is not exhaustive; the goal is to cover tools commonly encountered in these contexts rather than to provide a complete survey of the ecosystem. Scoring follows the baseline setup defined in Section 2.3.

Each simulator is evaluated by name and version where applicable. The matrix records a 0–2 score per criterion group and an applicability flag per task profile. Unconfirmed capabilities are marked unknown and treated conservatively as 0, as defined in Section 2.3.

#### 3.2. Simulator Platform Characterisation

Before applying the scoring matrix, the candidate platforms are briefly characterised to clarify their role in robotics, automation, and control workflows. This characterisation is intentionally compact: its purpose is not to replace detailed software documentation but to help readers identify the typical use context, integration profile, and practical constraints of each simulator before interpreting the comparative scores. Table 2 summarises the main profile of each platform considered in the analysis.

**Table 2.** Qualitative characterisation of the candidate simulation platforms considered in the global analysis.

| Platform         | Platform Type and Typical Use                                                                                                               | Robotics and Automation Integration Profile                                                                                             | Main Practical Profile                                                                                                                       |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| Gazebo           | Open-source robotics simulator for ROS/ROS 2, mobile robotics, manipulation, and academic prototyping.                                      | Strong fit for URDF/SDF workflows, ROS/ROS 2 integration, sensor simulation, plugins, and controller-oriented experiments.              | Flexible and widely adopted, but setup, middleware configuration, graphics dependencies, and contact tuning can increase complexity.         |
| MATLAB/Simulink  | Commercial model-based engineering environment for control design, multibody modelling, and system-level simulation.                        | Supports URDF/CAD-based import through Simscape Multibody, block-based modelling, solver configuration, and MATLAB toolbox integration. | Centralised and strong for control workflows, but dependent on proprietary licensing and less natural for ROS-native pipelines.              |
| NVIDIA Isaac Sim | GPU-oriented simulation framework based on NVIDIA Omniverse for photorealistic robotics, synthetic data, robot learning, and digital twins. | Supports rich rendering, sensor simulation, domain randomisation, ROS/ROS 2 connectivity, and learning-oriented pipelines.              | Powerful for perception-heavy and GPU-accelerated workflows, but hardware requirements and software-stack complexity are high.               |
| CoppeliaSim      | General-purpose robot simulation environment with scene modelling, scripting, motion planning support, and multiple APIs.                   | Supports scenario automation, robot modelling, control experiments, and ROS/ROS 2 interfaces depending on configuration.                | Useful for flexible prototyping, education, and multi-robot scenarios, but integration depth depends on plugins and middleware setup.        |
| Webots           | Open-source, multi-platform robot simulator used in education, research, and prototyping.                                                   | Supports robot modelling, sensors, physics-based simulation, controller programming, and ROS 2 integration.                             | Accessible and portable, especially for teaching and academic use, but less specialised for photorealistic or industrial workflows.          |
| MuJoCo           | Open-source physics engine for fast articulated-body simulation, contact dynamics, control, optimisation, and reinforcement learning.       | Well suited to dynamics-heavy and learning-oriented workloads where efficient contact simulation and programmatic control are central.  | Efficient and strong for research, but less complete as a full robotics middleware, sensor, and industrial simulation environment.           |
| PyBullet         | Python-accessible physics simulation environment used for robotics prototyping, machine learning, and reinforcement learning.               | Supports URDF/SDF model loading, dynamics, kinematics, collision queries, and programmatic simulation control through Python.           | Lightweight and scriptable, but visual realism, sensor realism, and industrial workflow integration are comparatively limited.               |
| RoboDK           | Commercial simulation and offline programming platform for industrial robot arms and manufacturing applications.                            | Focused on robot-cell simulation, CAD/CAM integration, path generation, post-processing, and offline program generation.                | Practical for industrial automation and OLP, but not designed primarily for contact-rich physics, sensor realism, or ROS research workflows. |
| ABB RobotStudio  | Vendor-specific industrial simulation and offline programming environment for ABB robotic systems and production cells.                     | Supports modelling, programming, verification, and commissioning within the ABB robot-controller ecosystem.                             | Strong for ABB deployments and training, but less general-purpose and less suited to heterogeneous open research workflows.                  |

Note: This table provides a compact platform-level characterisation. Detailed feature-level comparison is addressed separately in the scoring matrix and comparative capability tables.

### 3.3. Comparative Feature Matrix

To complement the qualitative platform characterisation, Tables 3 and 4 provide a compact feature-level comparison of the candidate simulators. The objective is to allow readers to inspect the main capabilities of each platform independently of the weighted scoring procedure. The comparison is qualitative and should be interpreted as a baseline

assessment under typical use conditions, since support may depend on simulator version, plugins, operating system, hardware, licensing options, and workflow configuration.

The qualitative classifications were assigned using the same evidence sources employed in the scoring matrix, including literature review, official documentation, and practical workflow observations. Consequently, the feature matrices should be interpreted as comparative decision-support tools rather than as exhaustive or absolute assessments of simulator capabilities.

The symbols used in the tables are defined as follows: “++” indicates strong or native support; “+” indicates supported functionality; “~” indicates partial, indirect, or configuration-dependent support; and “-” indicates limited support or that the feature is not a primary use case of the platform.

**Table 3.** Technical feature comparison of the candidate simulation platforms.

| Platform            | ROS/<br>ROS 2 | URDF/<br>SDF | Physics/<br>Contact | Sensors | Visual<br>Realism | GPU/<br>Headless | RL | Digital<br>Twins | OLP | API/<br>Scripting |
|---------------------|---------------|--------------|---------------------|---------|-------------------|------------------|----|------------------|-----|-------------------|
| Gazebo              | ++            | ++           | +                   | +       | +                 | +                | ~  | ~                | -   | ++                |
| MATLAB/<br>Simulink | ~             | +            | ++                  | +       | +                 | ~                | ~  | ~                | ~   | +                 |
| NVIDIA<br>Isaac Sim | +             | +            | ++                  | ++      | ++                | ++               | ++ | ++               | ~   | ++                |
| CoppeliaSim         | +             | +            | +                   | +       | +                 | ~                | ~  | ~                | ~   | ++                |
| Webots              | +             | +            | +                   | +       | +                 | +                | ~  | ~                | -   | +                 |
| MuJoCo              | -             | +            | ++                  | -       | ~                 | +                | ++ | -                | -   | ++                |
| PyBullet            | -             | ++           | +                   | ~       | ~                 | +                | ++ | -                | -   | ++                |
| RoboDK              | -             | ~            | -                   | -       | +                 | ~                | -  | +                | ++  | +                 |
| ABB                 | -             | -            | -                   | -       | +                 | ~                | -  | +                | ++  | +                 |
| RobotStudio         |               |              |                     |         |                   |                  |    |                  |     |                   |

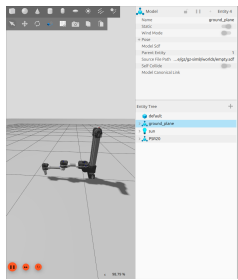
Note: “++” = strong/native support; “+” = supported; “~” = partial, indirect, or workflow-dependent support; “-” = limited support or not a primary use case. The GPU/headless column jointly summarises GPU acceleration, GPU-oriented execution, and support for non-graphical or headless workflows. OLP = offline programming.

**Table 4.** Practical suitability comparison of the candidate simulation platforms.

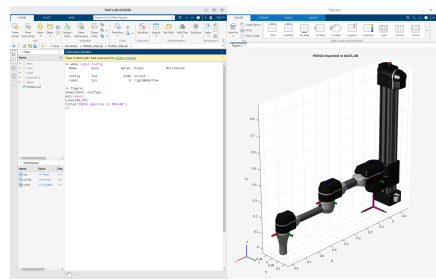
| Platform            | Setup Effort | Licensing Profile            | Learning Curve | Teaching | Research | Industry |
|---------------------|--------------|------------------------------|----------------|----------|----------|----------|
| Gazebo              | High         | Open-source                  | High           | +        | ++       | +        |
| MATLAB/<br>Simulink | Medium       | Commercial                   | Medium         | +        | +        | ++       |
| NVIDIA Isaac<br>Sim | High         | Free/commercial<br>ecosystem | High           | ~        | ++       | ++       |
| CoppeliaSim         | Medium       | Mixed licensing              | Medium         | +        | +        | ~        |
| Webots              | Low/Medium   | Open-source                  | Low/Medium     | ++       | +        | +        |
| MuJoCo              | Low/Medium   | Open-source                  | Medium         | ~        | ++       | ~        |
| PyBullet            | Low          | Open-source                  | Low/Medium     | +        | ++       | ~        |
| RoboDK              | Medium       | Commercial                   | Medium         | +        | ~        | ++       |
| ABB                 | Medium/High  | Vendor commercial            | Medium/High    | +        | ~        | ++       |
| RobotStudio         |              |                              |                |          |          |          |

Note: Practical suitability is qualitative and context-dependent. Teaching, research, and industry columns use the same symbol convention as Table 3.

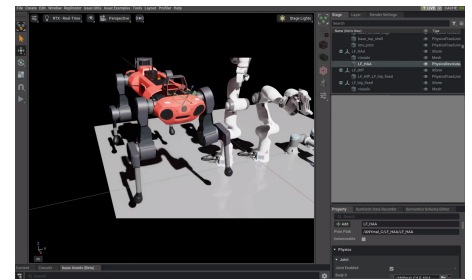
To complement the tabular comparison, Figure 2 provides representative visual examples of the graphical interfaces or simulation environments associated with the analysed platforms. The figure is intended as a visual aid for readers from applied engineering contexts rather than as an additional evaluation criterion.



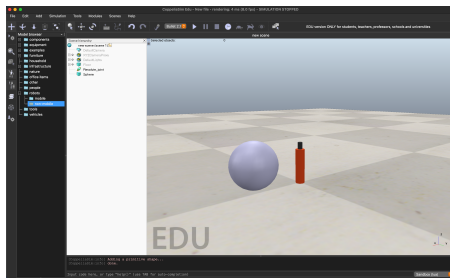
(a) Gazebo



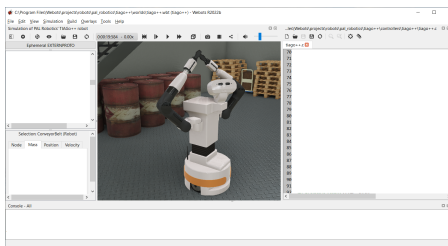
(b) MATLAB/Simulink



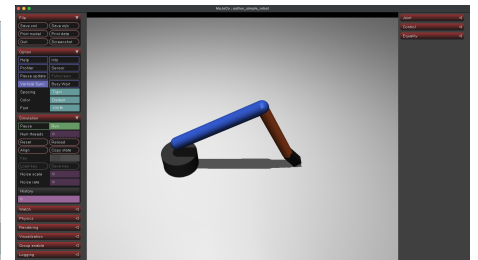
(c) NVIDIA Isaac Sim



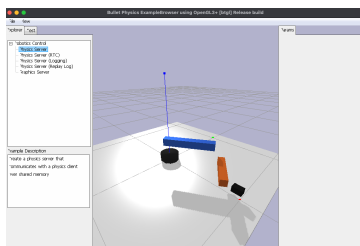
(d) CoppeliaSim



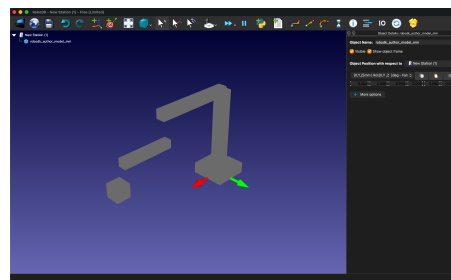
(e) Webots



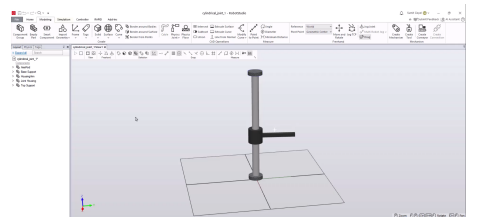
(f) MuJoCo



(g) PyBullet



(h) RoboDK



(i) ABB RobotStudio

**Figure 2.** Representative visual examples of the analysed robotics simulation platforms: (a) Gazebo, original screenshot generated by the authors during the case-study workflow; (b) MATLAB/Simulink, original screenshot generated by the authors during the case-study workflow; (c) NVIDIA Isaac Sim 4.0, adapted from the NVIDIA Technical Blog [20]; (d) CoppeliaSim, original screenshot generated by the authors using CoppeliaSim Edu with a custom scene composed of basic geometric primitives; (e) Webots, adapted from visual material available in the official Webots repository, distributed under the Apache License 2.0 [21,22]; (f) MuJoCo, original screenshot generated by the authors using a custom MJCF model composed of basic geometric primitives; (g) PyBullet, original screenshot generated by the authors using a custom scene composed of basic geometric primitives; (h) RoboDK, original screenshot generated by the authors using the official RoboDK desktop environment with a custom geometric model created by the authors; and (i) ABB RobotStudio, adapted from the official ABB RobotStudio tutorial page [23]; the source does not identify the specific software version represented in the screenshot. The figure is included only as an illustrative visual aid to support platform identification and does not constitute an evaluation criterion. The colours shown are native to the respective software interfaces and scene visualisations and do not encode scientific or comparative information. Permission requirements for third-party visual material, where applicable, are handled with the corresponding rights holders. All product names, trademarks, software interfaces, screenshots, and visual materials remain the property of their respective owners.

### 3.4. Scoring Matrix and Capability Summary

Table 5 summarises how the candidate simulators align with the six criterion groups defined in Table 1. The 0–2 scores provide a compact comparative view across platforms, with conservative scoring applied when capabilities cannot be reliably confirmed.

**Table 5.** Preliminary 0–2 scoring summary across the candidate simulators and criterion groups. Scores are conservative when capabilities cannot be reliably confirmed.

| Simulator        | Phys. | Sens./Vis. | Perf./Scale | Integration | Extens. | Practical |
|------------------|-------|------------|-------------|-------------|---------|-----------|
| Gazebo           | 1     | 1          | 1           | 2           | 2       | 2         |
| MATLAB/Simulink  | 2     | 2          | 1           | 1           | 1       | 1         |
| NVIDIA Isaac Sim | 2     | 2          | 2           | 1           | 2       | 1         |
| CoppeliaSim      | 1     | 1          | 1           | 1           | 2       | 2         |
| Webots           | 1     | 1          | 2           | 2           | 2       | 2         |
| MuJoCo           | 2     | 0          | 2           | 0           | 2       | 2         |
| PyBullet         | 1     | 0          | 2           | 0           | 2       | 2         |
| RoboDK           | 0     | 0          | 2           | 1           | 1       | 1         |
| ABB RobotStudio  | 0     | 0          | 2           | 1           | 1       | 1         |

**Legend:** Phys.: physical fidelity/contact modelling; Sens./Vis.: sensor modelling/visual realism; Perf./Scale: performance and scalability; Integration: ecosystem integration; Extens.: extensibility/programmability; Practical: licensing/cost, hardware footprint, portability, learning curve.

To improve the traceability of the scoring process, the evidence supporting the 0–2 score assignments is provided in Supplementary Table S1. This table records, for each simulator and criterion group, the assigned score, the main evidence type, and a concise rationale. The evidence combines literature sources, official documentation, reported simulator capabilities, practical workflow observations, and expert judgement. The scores should therefore be interpreted as conservative decision-support assessments rather than absolute measurements of simulator quality. Supplementary Material S1 also provides a reusable framework checklist, a blank scoring matrix template, and a filled RL-headless example that can be duplicated and adapted in the Excel workbook with automatic calculation of weighted numerators and normalised scores.

Two main patterns emerge. First, high-fidelity and photorealistic platforms generally require more hardware and more complex workflows, whereas lightweight engines favour throughput but offer more limited sensor realism [1,3,4]. Second, ecosystem fit (e.g., ROS/URDF integration, industrial offline programming (OLP) support, or model-based workflows) often becomes decisive even when physics-related capabilities are strong [1,3].

### 3.5. Task-Oriented Recommendations

Table 6 presents a task-oriented shortlist linking common task intents to primary and secondary candidates [2–4,6].

**Table 6.** Task-oriented shortlist (primary/secondary) consistent with the guiding questions.

| Task Intent (Examples)                                                       | Primary Candidates          | Secondary Candidates         |
|------------------------------------------------------------------------------|-----------------------------|------------------------------|
| Robot control and simulation (ROS-centric, URDF)                             | Gazebo                      | Webots; MATLAB/Simulink      |
| Digital twin and industrial simulation (photorealism)                        | Isaac Sim                   | RoboDK; RobotStudio          |
| Machine learning and large-scale simulation (batched reinforcement learning) | MuJoCo; PyBullet            | Isaac Sim                    |
| Introductory robotics teaching                                               | Webots                      | CoppeliaSim; Gazebo          |
| Undergraduate academic simulation                                            | Webots; Gazebo              | MATLAB/Simulink; CoppeliaSim |
| Master-level robotics, automation, and control workflows                     | Gazebo; MATLAB/Simulink     | Webots; Isaac Sim            |
| Doctoral and advanced research simulation                                    | MuJoCo; PyBullet; Isaac Sim | Gazebo                       |
| Technical and industrial training                                            | RoboDK; RobotStudio         | MATLAB/Simulink; Gazebo      |
| Industrial automation and offline programming (OLP)                          | RoboDK; RobotStudio         | MATLAB/Simulink              |
| Mobile robotics simulation (navigation, sensors)                             | Gazebo; Webots              | CoppeliaSim                  |
| Multi-physics and model-based control co-design                              | MATLAB/Simulink             | Gazebo                       |

The educational and training rows distinguish between introductory teaching, undergraduate use, advanced academic research, and technical or industrial training. This distinc-

tion is relevant because accessible graphical environments may be preferable for early-stage learning, whereas ROS-oriented, model-based, reinforcement-learning, or vendor-specific tools become more appropriate as the learning objectives move towards advanced research, automation engineering, or industrial deployment.

### 3.6. Illustrative Weighted Ranking: RL-Headless Profile

To illustrate the aggregation rule in practice, this section applies the proposed weighting scheme to a reinforcement learning workflow executed under headless conditions [4,7]. In this profile, Performance/Scalability and Extensibility/Programmability are treated as critical ( $w = 3$ ), while Physical Fidelity and Practical Constraints are treated as important ( $w = 2$ ). Since the workflow is headless and does not require ROS integration, Sensor Modelling/Visual Realism and Ecosystem Integration are marked as N/A ( $w = 0$ ).

The resulting weight profile is

$$\begin{aligned} \text{Phys.} &= 2, & \text{Sens./Vis.} &= 0, & \text{Perf./Scale} &= 3, \\ \text{Integration} &= 0, & \text{Extens.} &= 3, & \text{Practical} &= 2. \end{aligned}$$

Using the aggregation formula defined in Section 2.5, the denominator for this profile is

$$2 \times (2 + 3 + 3 + 2) = 20.$$

For each simulator, the weighted score is obtained by multiplying each applicable criterion score by the corresponding priority weight, summing the weighted values, and normalising the result by the maximum attainable score for the selected profile.

Table 7 applies this weighting profile to the 0–2 criterion-group scores reported in Table 5, producing a task-aligned ranking. Criteria marked as N/A do not contribute to the final score, so the ranking reflects this reinforcement learning workflow rather than a universal best-simulator ordering.

**Table 7.** RL-headless weighted scores (ordered in descending order) computed from Table 5 using the task-aligned weighting profile.

| Rank | Simulator        | Computation (Numerator)                                  | Score          |
|------|------------------|----------------------------------------------------------|----------------|
| 1    | MuJoCo           | $2 \times 2 + 3 \times 2 + 3 \times 2 + 2 \times 2 = 20$ | $20/20 = 1.00$ |
| 2    | PyBullet         | $2 \times 1 + 3 \times 2 + 3 \times 2 + 2 \times 2 = 18$ | $18/20 = 0.90$ |
| 3    | Webots           | $2 \times 1 + 3 \times 2 + 3 \times 2 + 2 \times 2 = 18$ | $18/20 = 0.90$ |
| 4    | NVIDIA Isaac Sim | $2 \times 2 + 3 \times 2 + 3 \times 2 + 2 \times 1 = 18$ | $18/20 = 0.90$ |
| 5    | Gazebo           | $2 \times 1 + 3 \times 1 + 3 \times 2 + 2 \times 2 = 15$ | $15/20 = 0.75$ |
| 6    | CoppeliaSim      | $2 \times 1 + 3 \times 1 + 3 \times 2 + 2 \times 2 = 15$ | $15/20 = 0.75$ |
| 7    | MATLAB/Simulink  | $2 \times 2 + 3 \times 1 + 3 \times 1 + 2 \times 1 = 12$ | $12/20 = 0.60$ |
| 8    | RoboDK           | $2 \times 0 + 3 \times 2 + 3 \times 1 + 2 \times 1 = 11$ | $11/20 = 0.55$ |
| 9    | ABB RobotStudio  | $2 \times 0 + 3 \times 2 + 3 \times 1 + 2 \times 1 = 11$ | $11/20 = 0.55$ |

Under this profile, MuJoCo achieves the highest normalised score, followed by a second tier formed by PyBullet, Webots, and NVIDIA Isaac Sim. Gazebo and CoppeliaSim follow closely, while MATLAB/Simulink ranks above the industrial OLP-oriented tools but below the more programmatic and learning-oriented simulation environments. This ranking should be interpreted as an illustration of the proposed weighting procedure for the selected criterion groups, rather than as a complete assessment of reinforcement-learning ecosystem maturity. It illustrates how the same scoring matrix can produce different rankings under different weighting profiles.

## 4. Illustrative Case Study: URDF-Based Gazebo and MATLAB/Simulink

### 4.1. Case Study Scope and Simulator Selection

The purpose of this case study is to illustrate how the proposed framework captures practical workflow factors in a representative automation and control scenario. The selected case study involves the comparison of two different simulator profiles: Gazebo was selected as a representative open-source ROS/URDF-oriented robotics simulator, while MATLAB/Simulink was selected as a representative commercial model-based simulation environment [2,18,19,24].

The framework was therefore examined through a URDF-based workflow implemented in both Gazebo and MATLAB/Simulink. Both environments used the PSR20 robot as a common basis, with platform-specific URDF variants adapted to each environment's workflow conventions. This kept the comparison focused on practical workflow differences rather than on robot geometry.

In Gazebo, the model followed a ROS 2-oriented URDF/Xacro workflow, with Xacro used to generate structured robot descriptions, together with supporting launch files and structural verification in RViz [10,25]. In MATLAB/Simulink, the corresponding URDF variant was imported and verified as a structured articulated model [18,19].

The case study covers model import, structural verification, representative joint-motion execution, setup conditions, toolchain preparation, learning curve, runtime behaviour, and workflow-level computational demand. Computational demand is assessed through instrumented measurements of the tested representative motion sequence, including observed execution interval, CPU utilisation, RAM usage, GPU utilisation, and VRAM allocation. These measurements are used as case-study evidence rather than as a general simulator benchmark. No claims are made regarding sensor realism, contact-rich interaction, physical accuracy, real-time factor, or sim-to-real transfer performance.

The comparison is organised around practical indicators aligned with the framework: setup conditions and toolchain preparation, workflow interaction and integration overhead, computational requirements, runtime behaviour, and practical constraints. Platform-specific constraints, including proprietary tooling requirements where applicable, are explicitly noted.

### 4.2. Setup Conditions and Toolchain Preparation

Gazebo required substantially more setup effort. A virtual machine attempt was not viable, as the environment became unstable before the URDF workflow could be used reliably. An earlier attempt on macOS was also unsuccessful, since Gazebo could not start reliably under the available graphics conditions. These difficulties motivated the adoption of a native Ubuntu dual-boot setup to obtain direct access to hardware resources.

The documented workflow ran on a Lenovo ThinkPad P16v Gen 2 with an Intel Core Ultra 9 185H, 96 GB RAM, and an NVIDIA RTX 3000 Ada Generation Laptop GPU. This represents a favourable execution platform rather than a minimum requirement. Results on weaker systems may differ, particularly in graphics-dependent workflows where setup friction and runtime robustness are harder to guarantee.

Even on native Linux, Gazebo required assembling a broader stack: ROS 2 Jazzy, Gazebo Harmonic, graphics support, and ROS–Gazebo communication via `ros_gz` [10,25]. The URDF workflow also required small platform-specific adjustments before simulation startup became reliable. These observations describe setup and toolchain preparation constraints rather than a direct limitation of Gazebo's simulation capabilities.

MATLAB/Simulink required a more contained setup. With an academic license and the required toolboxes available, the model could be imported and configured within the same environment [18,19]. In the documented trial, a first functional model with motion

was obtained in approximately 30 min, with no file-level adjustments to the URDF and less dependence on operating-system-specific preparation than Gazebo. Cross-platform reproducibility was not benchmarked.

#### 4.3. Workflow Interaction, Integration Overhead, and Learning Curve

A similar contrast was observed in the learning curve. Gazebo demanded attention to operating-system issues, graphics dependencies, middleware installation, workspace organisation, launch logic, URDF integration, and visualisation tooling. MATLAB/Simulink was more self-contained, although enabling motion still required platform-specific knowledge, namely joint actuation setup, Simulink-PS conversion blocks, and solver-compatible signal handling. Under the documented conditions, Gazebo exhibited the steeper practical learning curve, while MATLAB/Simulink offered a faster path to a first runnable result.

The two platforms also differed in how interaction with the workflow was organised during iteration. In Gazebo, execution was distributed across multiple components: terminal-based launch commands, ROS 2 processes, simulator startup, scripted motion triggering, RViz-based verification, and rebuild steps whenever relevant packages changed. This is consistent with a robotics-oriented software stack, but it adds operational overhead during testing and debugging.

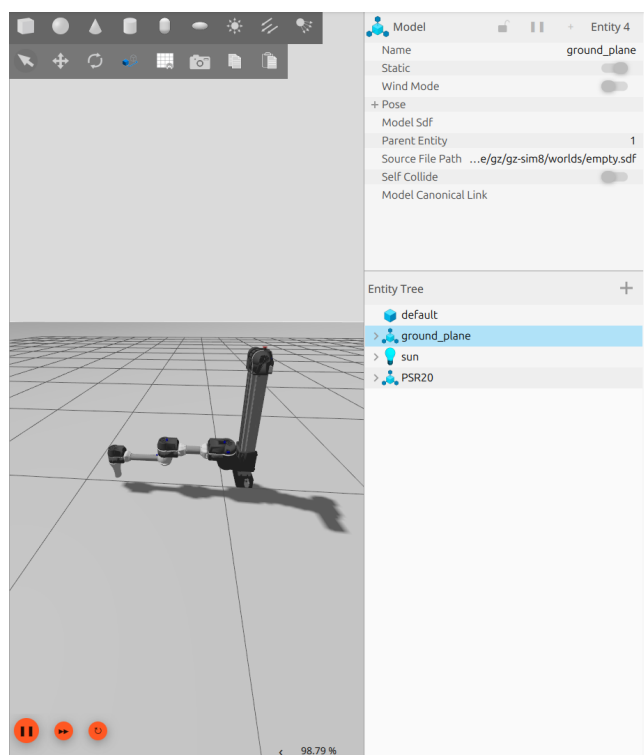
MATLAB/Simulink was more centralised. Scripting, model execution, workspace inspection, and visualisation were all available within the same environment, which reduced context switching during development. Under the present case-study conditions, this made initial experimentation and iterative testing more direct than in Gazebo, even though both environments ultimately worked with the same robot model.

Figures 3 and 4 illustrate the contrast between the two interaction models and highlight how the same URDF-based case study led to different workflow organisations in Gazebo/ROS 2 and MATLAB/Simulink.

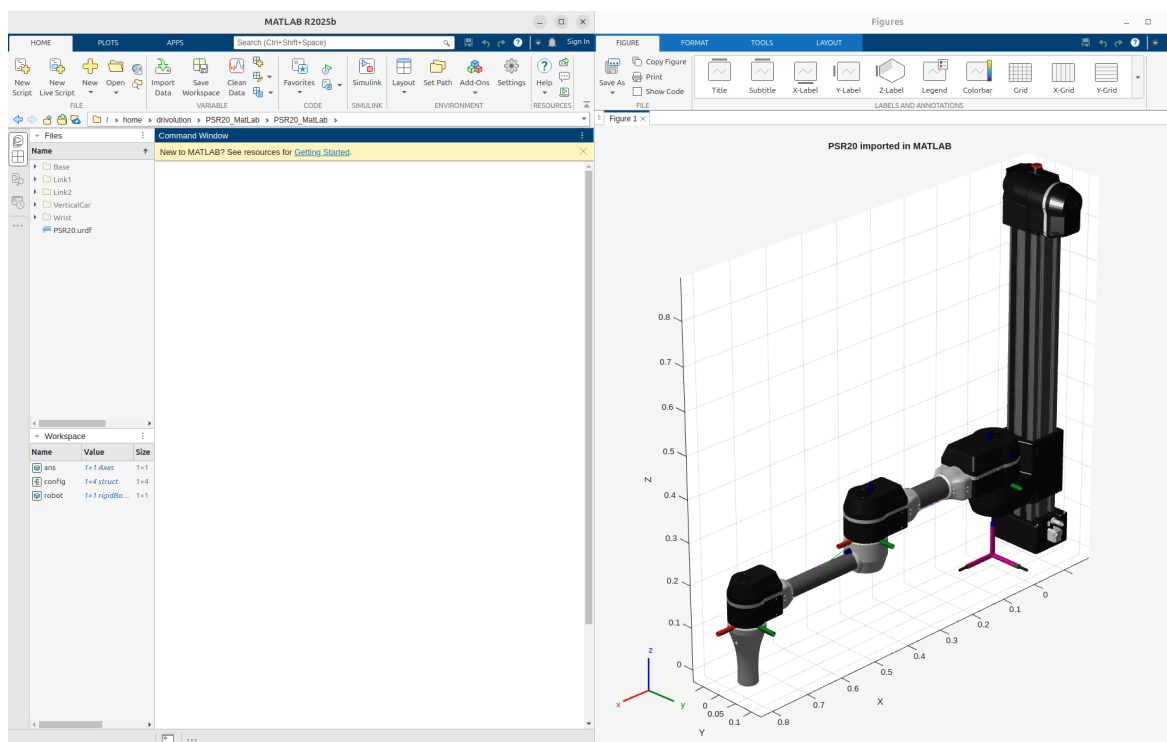
#### 4.4. Instrumented Runtime Metrics and Runtime Behaviour

Computational requirements and runtime behaviour were assessed through an instrumented execution of a representative motion sequence in both environments. The objective was not to establish a general simulator benchmark but to quantify the resource profile of the documented workflow under the specific experimental conditions used in this case study. The monitored metrics included observed execution interval, CPU utilisation, RAM usage, GPU utilisation, and VRAM allocation.

The representative motion sequence was defined as an equivalent joint-space sequence following the trajectory Home–Pose B–Pose C–Pose B–Home, with a nominal motion time of 9.5 s. The equivalent joints were Car\_to\_Link1 and Link1\_to\_Link2. The joint targets were Home ( $j_1 = 0.0, j_2 = 0.0$ ), Pose B ( $j_1 = 0.5, j_2 = 0.0$ ), Pose C ( $j_1 = 0.5, j_2 = -1.0$ ), Pose B ( $j_1 = 0.5, j_2 = 0.0$ ), and Home ( $j_1 = 0.0, j_2 = 0.0$ ). The nominal motion time corresponds only to the planned duration of the joint-space command sequence. It does not include simulator launch, model loading, toolchain initialisation, graphical backend overhead, middleware communication, monitoring overhead, or solver-dependent execution delays. The observed execution interval reported below therefore represents the wall-clock duration of the monitored workflow segment under each environment, which can exceed the nominal motion time, especially when software rendering, synchronisation, or toolchain overheads are significant.



**Figure 3.** Gazebo/ROS 2 workflow view for the PSR20 case study. The view shows the simulated PSR20 model in Gazebo together with the ROS 2-oriented execution context, where simulation, ROS 2 processes, command triggering, and verification are distributed across the robotics software stack. The colours shown are native to the Gazebo and ROS 2 interfaces and do not encode scientific or comparative information.



**Figure 4.** MATLAB/Simulink workflow view for the PSR20 case study. The view shows the imported PSR20 model and the MATLAB workspace/visualisation context, where scripting, workspace inspection, model import, and visualisation are available within a more centralised model-based engineering interface.

Table 8 summarises the measurement protocol. In Gazebo/ROS 2, the monitored interval covered the deterministic representative motion sequence after the Gazebo/ROS 2 stack had been launched and after a fixed simulator wait period; therefore, simulator and middleware startup time was excluded from the reported motion-window measurements. The monitored process scope included the Gazebo GUI, server, ROS 2 launch process, bridge, and robot state publisher. In MATLAB/Simulink, the measurement window was synchronised with the external monitor during the corresponding deterministic motion sequence, and the root MATLAB process and child processes were monitored recursively. The MATLAB session was executed with software OpenGL enabled, making the observed execution essentially CPU-bound under the documented conditions.

GPU utilisation and VRAM allocation were monitored through NVIDIA NVML as device-level measurements. Therefore, GPU-related values should not be interpreted as direct evidence of simulator-specific GPU compute workload. CPU utilisation is reported in normalised form by dividing the total monitored process CPU percentage by the 22 logical CPUs available on the test machine.

**Table 8.** Measurement protocol used for the instrumented representative motion sequence.

| Aspect              | Gazebo/ROS 2                                                                       | MATLAB/Simulink                                                                 |
|---------------------|------------------------------------------------------------------------------------|---------------------------------------------------------------------------------|
| Valid runs          | 3                                                                                  | 3                                                                               |
| Sampling interval   | 0.5 s                                                                              | 0.5 s                                                                           |
| Measurement window  | After Gazebo/ROS 2 startup and a fixed simulator wait period; motion sequence only | Synchronised with the external monitor during the deterministic motion sequence |
| CPU/RAM             | Gazebo/ROS 2 workflow processes monitored                                          | Root MATLAB process and child processes monitored recursively                   |
| GPU/VRAM            | NVIDIA NVML device-level values                                                    | NVIDIA NVML device-level values                                                 |
| Execution condition | Gazebo GUI, server, ROS 2 launch, bridge, and robot state publisher included       | MATLAB executed with software OpenGL enabled                                    |

The aggregated measurements in Table 9 show distinct resource profiles for the documented workflow. Gazebo/ROS 2 presented a shorter and highly repeatable observed execution interval, with low normalised CPU utilisation and memory usage below 1 GB. MATLAB/Simulink showed a substantially longer observed execution interval and higher CPU and RAM usage under the documented software OpenGL configuration. GPU utilisation was negligible in both environments, and the constant VRAM allocation indicates graphics-context allocation rather than sustained GPU compute workload. These results should therefore be interpreted as workflow-specific measurements for the tested representative motion sequence, not as a general performance ranking between the two simulators.

Accordingly, the values in Tables 8 and 9 should be interpreted as workflow-level observations under the reported setup. They are not intended as a general cross-platform performance benchmark, since the two environments differ in execution model, monitored process scope, graphical backend, and toolchain structure. Broader benchmark claims would require additional task categories, larger repetition counts, headless configurations, real-time factor measurements, trajectory-tracking accuracy assessment, contact-rich interaction tests, and sensor-realism evaluation.

**Table 9.** Aggregated runtime and resource measurements for the instrumented representative motion sequence. Values are reported as mean  $\pm$  standard deviation over three valid executions.

| Metric                          | Gazebo/ROS 2     | MATLAB/Simulink    |
|---------------------------------|------------------|--------------------|
| Observed execution interval (s) | 18.28 $\pm$ 0.01 | 167.28 $\pm$ 13.12 |
| Number of samples               | 35 $\pm$ 0       | 328 $\pm$ 25       |
| CPU mean, normalised (%)        | 5.55 $\pm$ 0.12  | 43.91 $\pm$ 0.88   |
| CPU peak, normalised (%)        | 6.68 $\pm$ 0.15  | 66.32 $\pm$ 1.91   |
| RAM mean (MB)                   | 943.5 $\pm$ 0.6  | 5134.6 $\pm$ 24.1  |
| RAM peak (MB)                   | 952.1 $\pm$ 0.7  | 5224.2 $\pm$ 51.6  |
| GPU mean utilisation (%)        | 0.00 $\pm$ 0.00  | 0.01 $\pm$ 0.02    |
| GPU peak utilisation (%)        | 0.00 $\pm$ 0.00  | 3.33 $\pm$ 5.77    |
| VRAM mean allocation (MB)       | 395.8 $\pm$ 0.0  | 395.8 $\pm$ 0.0    |
| VRAM peak allocation (MB)       | 395.8 $\pm$ 0.0  | 395.8 $\pm$ 0.0    |

Runtime behaviour was directly observable in both environments. In Gazebo, the robot remained stable during static observation, and the simplified dynamic sequence became reproducible after local adjustments to velocity limits and joint dynamic parameters such as damping and friction. Graphics-related `libEGL` warnings were observed, but they did not prevent simulator launch, robot spawning, or successful execution of the tested scenarios. In MATLAB/Simulink, the first valid execution already produced stable continuous motion driven by input signals, with no reported oscillatory behaviour or graphics-related failures. Under the tested motion-oriented conditions, Gazebo required more tuning to obtain a clean and repeatable dynamic demonstration, whereas MATLAB/Simulink provided a more immediately controlled execution context.

#### 4.5. Overall Practical Assessment

Taken together, the observations identify two distinct workflow profiles. Gazebo required a distributed ROS 2/Gazebo workflow involving operating-system preparation, middleware configuration, simulator startup, external command triggering, and visualisation tooling. MATLAB/Simulink provided a more centralised environment for model import, scripting, simulation, workspace inspection, and visualisation.

This does not imply that setup difficulty should be interpreted as a direct limitation of simulator capability. Rather, it shows that practical constraints, toolchain preparation, and workflow overhead are relevant selection factors that should be considered separately from physics-related or runtime capabilities.

Within the documented case-study scope, the case study is most informative for setup conditions, workflow interaction, integration overhead, practical constraints, learning curve, runtime behaviour, and workflow-level computational demand. The instrumented measurements provide quantitative evidence for the tested representative motion sequence, particularly regarding observed execution interval, CPU utilisation, RAM usage, GPU utilisation, and VRAM allocation. However, these measurements remain limited to a single motion-oriented scenario and do not support broad conclusions regarding sensor modelling, contact-rich interaction, physical accuracy, real-time factor, or sim-to-real transfer performance.

## 5. Discussion

The results support the main premise of this work: simulator selection benefits from a structured, task-oriented process rather than from familiarity, availability, or isolated feature comparisons [1–3]. The global comparison and the illustrative weighted ranking showed that simulator preference shifts with task priorities. This reinforces the value of the framework as a lightweight and traceable decision aid that links simulator choice

to explicit workflow requirements instead of treating simulator selection as a universal ranking problem.

The case study adds a practical dimension by showing that simulator selection in automation and control is not determined by physics-related capabilities alone. The comparison between a distributed robotics-oriented workflow and a centralised model-based workflow showed that setup conditions, integration overhead, interaction model, learning curve, and platform dependence can materially affect simulator suitability. These factors are often secondary in feature-based comparisons, but they can become decisive in real engineering workflows. In this sense, ecosystem fit and practical constraints should be considered part of the selection problem, not merely implementation details.

However, the proposed framework also has important limitations. First, the scoring matrix relies on an ordinal 0–2 scale. This makes the method simple, transparent, and easy to apply, but it also compresses complex simulator capabilities into broad categories. A score of 2 does not necessarily mean that a simulator is optimal for every configuration of a criterion, and a score of 1 may cover different degrees of partial support. Therefore, the matrix should be interpreted as a decision-support instrument rather than as a precise measurement of simulator performance.

Second, score assignment necessarily involves expert judgement. Although the rubric, literature synthesis, documentation review, and conservative treatment of unconfirmed capabilities reduce arbitrariness, they do not eliminate subjectivity. Different evaluators may assign different scores depending on their experience, target hardware, software version, or interpretation of a task requirement. This is particularly relevant for criteria such as practical constraints, integration overhead, and learning curve, which are strongly affected by the deployment context.

Third, the empirical application remains limited. The case study was restricted to a URDF-based workflow comparing Gazebo and MATLAB/Simulink, with emphasis on model import, motion execution, setup conditions, workflow observations, and limited instrumented runtime measurements. It does not constitute a broad evaluation of the framework across multiple robots, simulators, operating systems, task categories, or user profiles. As a result, the case study should be understood as an illustrative application of the framework rather than as a general benchmark of simulator quality.

In addition, the empirical instrumentation was restricted to a limited set of runtime and resource indicators. The case study included controlled monitoring of observed execution interval, CPU utilisation, RAM usage, GPU utilisation, and VRAM allocation across three valid executions per environment for the representative motion sequence. However, these measurements capture workflow-level resource behaviour under the documented conditions rather than a general simulator benchmark. The experiment did not measure real-time factor, trajectory-tracking error, contact accuracy, solver sensitivity, headless execution, or per-process resource attribution. GPU and VRAM measurements were collected through NVIDIA NVML at the device level, so VRAM allocation should not be interpreted as evidence of sustained GPU computation. Broader performance claims would require additional scenarios, headless configurations, and larger repetition counts.

These limitations do not undermine the framework, but they define its appropriate use. The proposed method is most useful as an early-stage, task-oriented selection procedure that makes assumptions, priorities, and trade-offs explicit. It should not replace detailed benchmarking when quantitative guarantees are required, nor should its weighted rankings be interpreted independently of the task profile used to generate them.

Future work should therefore extend the framework in three directions: broader empirical assessment across additional simulators and task profiles; sensor-intensive and contact-rich case studies; and instrumented measurements of computational performance,

repeatability, and sim-to-real behaviour. More broadly, the field would benefit from unified multi-task benchmarks and joint metrics capable of capturing simulation fidelity, computational efficiency, and sim-to-real transfer within a common evaluation framework [2,6,7].

## 6. Conclusions

This paper proposed a task-oriented decision framework for selecting robotics simulators in automation and control workflows. The framework combines six criterion groups, four guiding questions, a lightweight checklist, and a weighted scoring procedure to connect simulator selection with explicit task requirements instead of relying on familiarity, availability, or generic simulator rankings.

The main practical value of the framework is that it makes the selection process more transparent and traceable. By separating physical fidelity, sensor and visual modelling, performance and scalability, ecosystem integration, extensibility, and practical constraints, the method helps users identify why a given simulator is more suitable for a specific workflow. This is particularly relevant in mechanical engineering, automation, and applied robotics, where the appropriate tool may differ depending on whether the task involves ROS-based control, model-based design, reinforcement learning, digital twins, teaching, or industrial offline programming.

The global comparison showed that no simulator dominates all criteria. Platforms such as Gazebo, Webots, MuJoCo, PyBullet, NVIDIA Isaac Sim, MATLAB/Simulink, RoboDK, and ABB RobotStudio address different engineering needs and impose different trade-offs. The illustrative weighted ranking demonstrated how the same baseline scoring matrix can lead to different recommendations when the task profile changes. In this sense, the proposed methodology can support real software-selection decisions by making assumptions, priorities, and trade-offs explicit before committing to a simulation platform.

The URDF-based case study reinforced the importance of considering practical workflow factors alongside simulator capabilities. Under the documented conditions, Gazebo/ROS 2 and MATLAB/Simulink differed not only in execution behaviour, but also in setup effort, toolchain preparation, interaction model, integration overhead, learning curve, and workflow-level computational demand. The instrumented measurements provided quantitative evidence for the tested representative motion sequence but should not be interpreted as a general benchmark between the two platforms.

The work has limitations. The scoring matrix uses a compact ordinal 0–2 scale and therefore simplifies complex simulator capabilities. Score assignment also involves expert judgement and may vary with software version, hardware configuration, operating system, licensing conditions, plugin availability, and user experience. In addition, the empirical application was limited to one URDF-based Gazebo/ROS 2 and MATLAB/Simulink case study without evaluating contact-rich manipulation, sensor realism, real-time factor, trajectory-tracking accuracy, headless execution, or sim-to-real transfer.

Future work should extend the framework across additional simulators, robot models, operating systems, and task profiles, including sensor-intensive, contact-rich, digital-twin, and reinforcement-learning scenarios. More detailed measurements of computational performance, repeatability, physical fidelity, and sim-to-real behaviour would further strengthen the framework, while unified multi-task benchmarks remain an important open need for the field [2,6,7].

Overall, the proposed framework provides a compact and practical decision-support tool for simulator selection in engineering contexts. It does not replace detailed benchmarking when quantitative guarantees are required, but it offers a structured starting point for comparing alternatives, documenting rationale, and selecting simulation software according to the requirements of a specific engineering task.

**Supplementary Materials:** The following supporting information can be downloaded at <https://www.mdpi.com/article/10.3390/engproc2026145008/s1>, Supplementary Material S1: framework checklist, blank scoring matrix template, filled example, and scoring-justification table, provided as a PDF file.

**Author Contributions:** Conceptualization, T.A.T.B.B., C.M.A.V., P.M.R.C. and A.M.S.C.; methodology, T.A.T.B.B., C.M.A.V. and P.M.R.C.; validation, T.A.T.B.B., C.M.A.V. and P.M.R.C.; investigation, T.A.T.B.B., C.M.A.V. and P.M.R.C.; writing—original draft preparation, T.A.T.B.B.; writing—review and editing, T.A.T.B.B., C.M.A.V., P.M.R.C. and A.M.S.C.; supervision, C.M.A.V., P.M.R.C. and A.M.S.C.; funding acquisition, C.M.A.V. and A.M.S.C. All authors have read and agreed to the published version of the manuscript.

**Funding:** The authors gratefully acknowledge the funding provided by the Foundation for Science and Technology (FCT) of Portugal, within the scope of the project of the Research Unit on Materials, Energy and Environment for Sustainability (proMetheus), Ref. UID/05975/2020, financed by national funds through the FCT/MCTES; and the funding provided within the scope of the “Agenda DRIVOLUTION: Transition to the Factory of the Future”, project no. C632394276-00466981 with operation code 02/C05-i01.02/2022.PC644913740-00000022, within the framework of the Agendas/Mobilizing Alliances for Reindustrialization, Notice no. 2022-C05i0101-02, project 23, of the Recovery and Resilience Plan (PRR) of Portugal.

**Data Availability Statement:** All data supporting the findings of this study are available within the article and its Supplementary Material S1. No additional publicly archived datasets were generated or analysed.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## Abbreviations

|       |                                   |
|-------|-----------------------------------|
| API   | Application Programming Interface |
| CAD   | Computer-Aided Design             |
| CPU   | Central Processing Unit           |
| GPU   | Graphics Processing Unit          |
| GUI   | Graphical User Interface          |
| NVML  | NVIDIA Management Library         |
| OLP   | Offline Programming               |
| RAM   | Random Access Memory              |
| RL    | Reinforcement Learning            |
| ROS   | Robot Operating System            |
| ROS 2 | Robot Operating System 2          |
| RViz  | ROS Visualization Tool            |
| SDK   | Software Development Kit          |
| SDF   | Simulation Description Format     |
| URDF  | Unified Robot Description Format  |
| VRAM  | Video Random Access Memory        |
| Xacro | XML Macros                        |

## References

1. Afzal, A.; Katz, D.S.; Le Goues, C.; Timperley, C.S. A Study on the Challenges of Using Robotics Simulators for Testing. *arXiv* **2020**, arXiv:2004.07368. [[CrossRef](#)]
2. Collins, J.; Chand, S.; Vanderkop, A.; Howard, D. A Review of Physics Simulators for Robotic Applications. *IEEE Access* **2021**, *9*, 51416–51431. [[CrossRef](#)]
3. Flores Gonzalez, J.M.; Coronado, E.; Yamanobe, N. ROS-Compatible Robotics Simulators for Industry 4.0 and Industry 5.0: A Systematic Review of Trends and Technologies. *Appl. Sci.* **2025**, *15*, 8637. [[CrossRef](#)]
4. Kaup, M.; Wolff, C.; Hwang, H.; Mayer, J.; Bruni, E. A Review of Nine Physics Engines for Reinforcement Learning Research. *arXiv* **2024**, arXiv:2407.08590. [[CrossRef](#)]

5. Cervera, E. Run to the Source: The Effective Reproducibility of Robotics Code Repositories. *IEEE Robot. Autom. Mag.* **2024**, *31*, 125–134. [CrossRef]
6. Kargar, S.M.; Yordanov, B.; Harvey, C.; Asadipour, A. Emerging Trends in Realistic Robotic Simulations: A Comprehensive Systematic Literature Review. *IEEE Access* **2024**, *12*, 191264–191287. [CrossRef]
7. Salvato, E.; Fenu, G.; Medvet, E.; Pellegrino, F.A. Crossing the Reality Gap: A Survey on Sim-to-Real Transferability of Robot Controllers in Reinforcement Learning. *IEEE Access* **2021**, *9*, 153171–153187. [CrossRef]
8. Tseeva, F.M.; Shogenova, M.M.; Senov, K.M.; Liana, K.V.; Bozieva, A.M. Comparative Analysis of Two Simulation Environments for Robots, Gazebo, and CoppeliaSim in the Context of Their Use for Teaching Students a Course in Robotic Systems modelling. In Proceedings of the 2024 International Conference “Quality Management, Transport and Information Security, Information Technologies” (QM&TIS&IT), Nalchik, Russia, 23–27 September 2024; pp. 186–189. [CrossRef]
9. Gazebo Sim Documentation. Gazebo Sim Documentation. Available online: <https://gazebo-sim.org/docs/latest/> (accessed on 25 May 2026).
10. Gazebo Sim Documentation. Spawn URDF. Available online: [https://gazebo-sim.org/docs/latest/spawn\\_urdf/](https://gazebo-sim.org/docs/latest/spawn_urdf/) (accessed on 3 April 2026).
11. Cyberbotics. Webots User Guide. Available online: <https://cyberbotics.com/doc/guide/index> (accessed on 25 May 2026).
12. NVIDIA. NVIDIA Isaac Sim 6.0.0 Release Notes. Available online: [https://docs.isaacsim.omniverse.nvidia.com/6.0.0/overview/release\\_notes.html](https://docs.isaacsim.omniverse.nvidia.com/6.0.0/overview/release_notes.html) (accessed on 25 May 2026).
13. Google DeepMind. MuJoCo Documentation. Available online: <https://mujoco.readthedocs.io/> (accessed on 25 May 2026).
14. Coumans, E.; Bai, Y. PyBullet Quickstart Guide. Available online: [https://raw.githubusercontent.com/bulletphysics/bullet3/master/docs/pybullet\\_quickstartguide.pdf](https://raw.githubusercontent.com/bulletphysics/bullet3/master/docs/pybullet_quickstartguide.pdf) (accessed on 25 May 2026).
15. Coppelia Robotics. CoppeliaSim User Manual. Available online: <https://manual.coppeliarobotics.com/> (accessed on 25 May 2026).
16. RoboDK. RoboDK Documentation: Basic Guide. Available online: <https://robodk.com/doc/en/Basic-Guide.html> (accessed on 25 May 2026).
17. ABB Robotics. RobotStudio Suite. Available online: <https://www.abb.com/global/en/areas/robotics/products/software/robotstudio-suite> (accessed on 25 May 2026).
18. MathWorks. Import URDF Models. Available online: <https://www.mathworks.com/help/sm/ug/urdf-import.html> (accessed on 3 April 2026).
19. MathWorks. Simscape Multibody Documentation. Available online: <https://www.mathworks.com/help/sm/> (accessed on 3 April 2026).
20. NVIDIA. Supercharge Robotics Workflows with AI and Simulation Using NVIDIA Isaac Sim 4.0 and NVIDIA Isaac Lab. Available online: <https://developer.nvidia.com/blog/supercharge-robotics-workflows-with-ai-and-simulation-using-nvidia-isaac-sim-4-0-and-nvidia-isaac-lab/> (accessed on 9 July 2026).
21. Cyberbotics Ltd. Webots: Open-Source Robot Simulator. Available online: <https://github.com/cyberbotics/webots> (accessed on 9 July 2026).
22. Cyberbotics Ltd. Webots License. Available online: <https://github.com/cyberbotics/webots/blob/master/LICENSE> (accessed on 9 July 2026).
23. ABB Robotics. RobotStudio Tutorials. Available online: <https://www.abb.com/global/en/areas/robotics/products/software/robotstudio-suite/robotstudio-tutorials> (accessed on 9 July 2026).
24. Liu, C.K.; Negrut, D. The Role of Physics-Based Simulators in Robotics. *Annu. Rev. Control Robot. Auton. Syst.* **2021**, *4*, 35–58. [CrossRef]
25. ROS 2 Documentation. URDF. Available online: <https://docs.ros.org/en/jazzy/Tutorials/Intermediate/URDF/URDF-Main.html> (accessed on 3 April 2026).

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.