2026 NDIA MICHIGAN CHAPTER
GROUND VEHICLE SYSTEMS ENGINEERING
AND TECHNOLOGY SYMPOSIUM
DIGITAL ENGINEERING / SYSTEMS ENGINEERING (DE/SE) TECHNICAL SESSION
AUG. 11-13, 2026 - NOVI, MICHIGAN

DESIGNING A DIGITAL TWIN–ENABLED REQUIREMENT VERIFICATION WORKFLOW WITHIN AN MBSE FRAMEWORK

Omar Zeki[1], Farid Sahebsara[1], Mohammadreza Torkjazi[1], Michael R. Hieb[1], and Ali K. Raz[1]

[1]C5I Center, Systems Engineering and Operations Research Department, George Mason University, Fairfax, VA

ABSTRACT

Verification of functional requirements in Model-Based Systems Engineering environments remains fragmented across heterogeneous tools and manual processes. This paper presents a digital twin–enabled workflow that supports automated requirement verification through integration of SysML models, executable simulation environments, and verification evaluation functions. Within this scope, the objective is to formalize a verification workflow that preserves architectural abstraction while enabling automated, traceable, and simulation-driven evaluation of functional requirements. The approach establishes a continuous digital thread that maintains traceability between requirements, system architecture, and verification outcomes.

The workflow is demonstrated using a differential-drive robotic platform, where sensor data availability and update rate verification are used as representative examples of digital twin-based functional requirement evaluation. Results illustrate the feasibility of incorporating digital twin-driven verification into model-centric engineering processes while maintaining consistent verification feedback within the system model. The demonstration produced both passing and failing verification outcomes, illustrating the workflow's ability to surface requirement-design mismatches.

Citation: O. Zeki, F. Sahebsara, M.Torkjazi, M. R. Hieb, A. K. Raz, "Designing a Digital Twin–Enabled Requirement Verification Workflow within an MBSE Framework," In *Proceedings of the Ground Vehicle Systems Engineering and Technology Symposium* (GVSETS), NDIA, Novi, MI, Aug. 11-13, 2026.

1. INTRODUCTION

Model-Based System Engineering (MBSE) has become a dominant paradigm for managing the complexity of modern Cyber-Physical Systems (CPS). By replacing document-centric artifacts with formal, machine-interpretable models, MBSE enhances traceability, consistency, and architectural rigor across the system lifecycle [1, 2]. Within this paradigm, requirements, structure, and behavior are captured in interconnected models that support systematic reasoning and change management.

A primary motivation for adopting MBSE is the earlier integration of Verification and Validation (V&V) activities. Model-based requirement verification techniques have demonstrated that formalized requirements can be linked to architectural and behavioral models to enable automated analysis [3, 4]. More recent frameworks incorporate traceability mechanisms and simulation workflows to support structured compliance assessment [5, 6]. Despite these advances, verification processes often remain decoupled from model evolution, instead of being continuously integrated with evolving system models throughout the lifecycle.

Concurrently, Digital Twins (DTws) have emerged as executable, physics-based representations capable of supporting dynamic system analysis [7, 8]. When integrated with MBSE, DTws offer a pathway to connect descriptive system architecture with runtime simulation. However, most reported efforts emphasize either architectural traceability or simulation execution, without defining a structured process that governs how requirements are evaluated through the DTw environment.

This work builds upon two foundational efforts. First, a modular physical–digital testbed architecture was developed to enable synchronized experimentation between physical and virtual domains within an MBSE framework. Second, the architectural role of MBSE as the backbone for federated DTw development and lifecycle traceability was established [9, 10]. Building on these foundations, this paper proposes a structured requirement verification workflow that formalizes the interaction between MBSE artifacts, a physics-based DTw, and automated functional requirement evaluation.

The remainder of this paper is organized as follows: Section 2 defines the problem and scope of the work, Section 3 presents the proposed requirement verification workflow using an MBSE-based DTw, Section 4 demonstrates the workflow through data availability and update rate verification, and Section 5 concludes the paper.

2. PROBLEM DEFINITION AND SCOPE

Although MBSE provides structured representations of requirements and architecture, and DTws enable executable system representations, their interaction remains insufficiently formalized for verification activities. In many current practices, architectural models and simulation environments operate in parallel rather than within a unified verification process, resulting in verification activities that are performed offline or loosely coupled to architectural artifacts.

This limitation stems from the abstraction gap between descriptive Systems Modeling Language (SysML) models and executable simulation models. Architectural models preserve abstraction to support system-level reasoning, whereas DTw environments require explicit physical parameters and runtime execution semantics. Without structured integration, traceability between requirements, architecture, simulation configuration, and verification evidence becomes fragmented.

Even when formal trace links exist in an MBSE environment, evaluation logic is often externalized to tool-specific scripts or post-processed analyses. This weakens continuity of the Digital Thread (DT) and limits automated, runtime verification.

Building on the architectural and experimental foundations established in prior work [9, 10], the remaining challenge is not the availability of modeling or simulation capabilities, but the absence of a clearly defined workflow governing how requirements are mapped to executable DTw

artifacts and evaluated systematically during runtime.

Accordingly, the scope of this paper is limited to defining a structured workflow for online verification of functional requirements using an MBSE-based DTw. The emphasis is on process formalization rather than requirement authoring techniques, non-functional verification, certification workflows, or operational lifecycle twins. Within this scope, the objective is to formalize a verification workflow that preserves architectural abstraction while enabling automated, traceable, and simulation-driven evaluation of functional requirements.

3. DIGITAL TWIN–ENABLED REQUIREMENT VERIFICATION WORKFLOW

This section presents the proposed workflow for automated requirement verification using a DTw within an MBSE environment. The workflow integrates system modeling, requirements management, and DTw-based verification through a coordinated Digital Engineering (DE) toolchain.

The objective is to maintain traceability between system requirements, architecture elements, and verification results while enabling automated evaluation of requirement compliance through executable system representations.

The proposed workflow is structured into four sequential phases:

1. Requirement specification and verification planning within the MBSE model
2. Requirements exchange and synchronization across engineering environments
3. DTw–based verification execution
4. Verification feedback and model update within the MBSE environment

Together these phases establish a continuous DT between requirements, models, and verification results.

Figure 1 illustrates the workflow as a SysML activity diagram with activities allocated across the MBSE-based DTw environment. Although the workflow is demonstrated using specific tools, the proposed approach is designed to be tool-agnostic in principle, as each phase relies on standard interfaces and exchange formats that can be supported by alternative modeling and DTw environments.

3.1. Requirement Specification and Verification Planning in the MBSE Model

The workflow begins in the MBSE environment, where system requirements and verification artifacts are defined using SysML. Requirements are modeled as $<< requirement >>$ elements organized hierarchically and include attributes such as identifier, description, rationale, and verification method.

System components subject to verification are modeled as $<< block >>$ elements within the system's logical architecture, representing the architectural units that implement requirements.

Verification planning is performed by defining test cases as $<< TestCase >>$ elements. Each test case captures its requirement trace, components under test, preconditions, execution steps, and expected results as structured text within the element's documentation property. Test cases are grouped according to verification domains and assigned to operational scenarios representing different system configurations.

Traceability is established through two relationships. Requirement–test

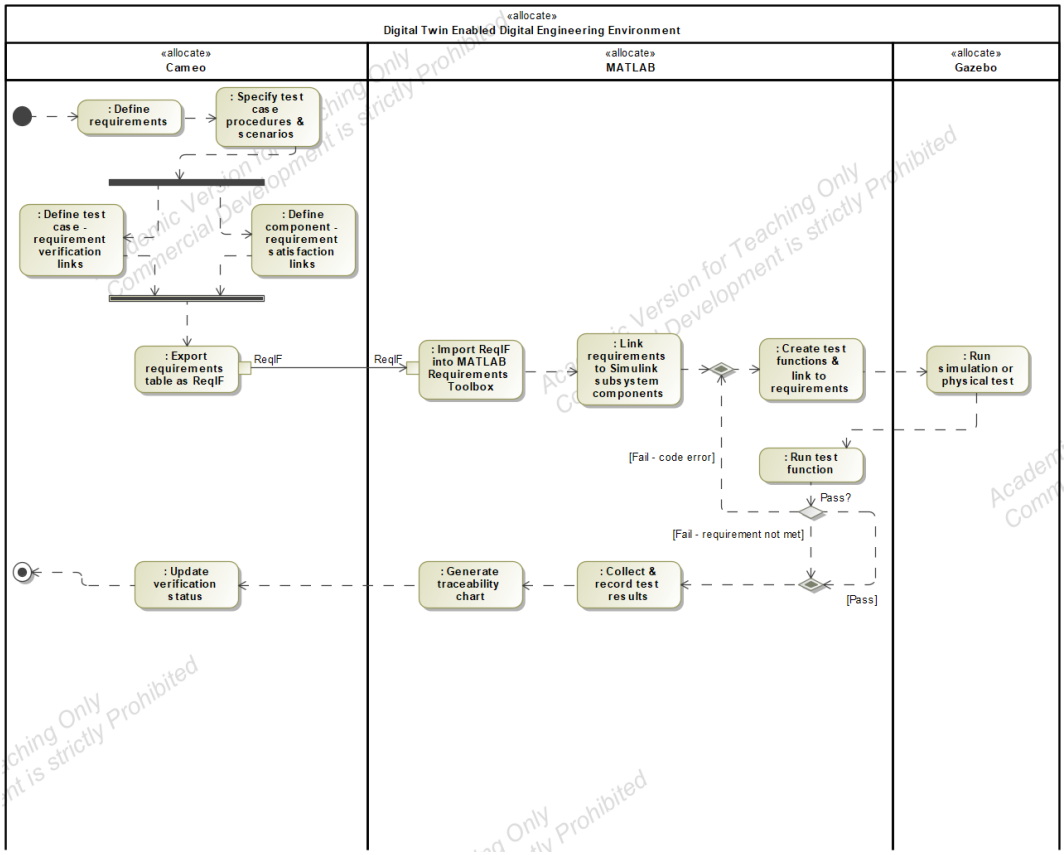

Figure 1: Digital twin–enabled requirement verification workflow.

relationships are modeled using the SysML $<< \ verify \ >>$ dependency, linking requirements to the test cases responsible for verification. Requirement–component relationships are defined using the $<< \ satisfy \ >>$ dependency, linking requirements to architectural elements implementing them. These relationships are captured in verification and satisfaction matrices within the MBSE environment.

3.2. Requirements Exchange and Synchronization

Requirements are transferred from the MBSE environment to the DTw environment using the Requirements Interchange Format (ReqIF), an Object Management Group (OMG) standard for tool-independent requirements exchange. ReqIF preserves requirement hierarchy, attributes, and traceability relationships during transfer [11, 12, 13].

In the implementation presented in this work, the requirement model is exported from the MBSE tool as a ReqIF file and imported into the DTw environment's requirements management framework. During import, requirement identifiers, descriptions, and hierarchical relationships are preserved, enabling consistent linkage between system models and verification artifacts used in simulation.

3.3. Digital Twin–Based Verification Execution

After import, each requirement is linked to the executable system model representing its implementation using an *Implemented By* relationship. These links connect requirements to model artifacts such as control algorithms, sensor interfaces, and actuator logic.

While ReqIF transfers requirement elements and their attributes between environments, the structured content of test cases authored within <<TestCase>> documentation properties is not preserved as executable artifacts in the exchange. Test case content therefore serves as the specification basis for two distinct activities performed in the DTw environment. First, preconditions inform the configuration of the DTw execution environment, defining which simulation scenario, system configuration, and tool interfaces must be active before a test runs. For the demonstration in this work, the shared precondition of test scenario TS2 establishes that the digital vehicle is running in Gazebo, MATLAB is connected via Robot Operating System (ROS2), and the vehicle is in an active operation scenario. Second, execution steps and expected results inform the implementation of each test function: the steps define what the function observes during simulation, and the expected results define the acceptance criteria encoded in the verification logic. This separation preserves the MBSE model as the authoritative source of test case specifications while allowing executable evaluation to live in the DTw environment.

Test functions are implemented as automated evaluation scripts and associated with requirements through *Verified By* relationships. These functions encode acceptance criteria and automatically evaluate simulation outputs.

The DTw execution environment provides a physics-based representation of system dynamics and behavior, enabling assessment of requirement compliance under representative operating conditions. After test and evaluation, verification functions evaluate the outputs to determine requirement compliance. A test may pass, fail due to system behavior, or fail due to errors in the verification implementation itself. This distinction ensures that recorded failures reflect actual requirement violations rather than issues in verification logic.

Verification results are collected and used to generate traceability views linking requirements, system components, verification functions, and test outcomes. These views provide visibility into verification coverage and help identify requirements lacking implementation or test validation.

4. WORKFLOW DEMONSTRATION

To demonstrate the proposed workflow, a subset of test cases was executed using the DTw of a TurtleBot3 Burger. The TurtleBot3 is a compact differential-drive mobile robot equipped with a LIDAR scanner, a 3-axis IMU, and wheel encoders, making it a suitable platform for evaluating sensor data availability and navigation-related requirements.

Following the workflow described in Section 3, two categories of functional requirements were authored in Cameo MagicDraw as SysML $<< \ requirement \ >>$ elements: data availability requirements (Req 2.1), which verify that each sensor data stream is active and accessible during DTw operation, and update rate requirements (Req 2.3), which verify that sensor and computed data meet minimum frequency thresholds. Test cases for both categories were specified and organized under test scenario TS2 (digital-only operation). Verification and satisfaction relationships were established in their respective matrices, and the requirement model was exported as a ReqIF file.

Upon import into the MATLAB Requirements Toolbox, each requirement was linked to its corresponding Simulink subsystem component via an *Implemented By* relationship. An illustrative implementation

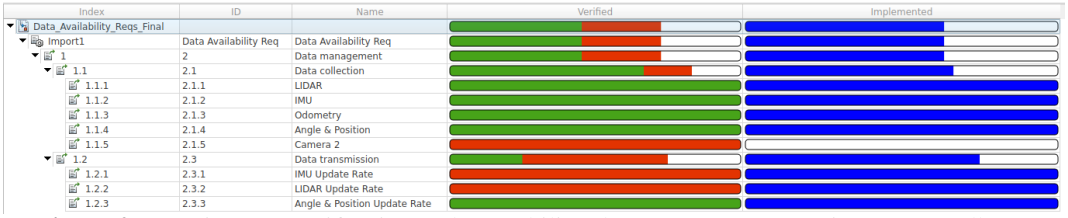

Figure 2: Requirement verification and traceability chart (MATLAB Requirements Toolbox).

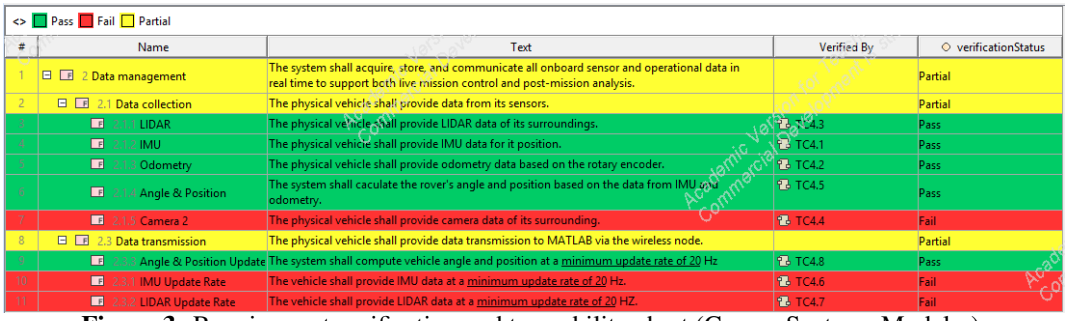

Figure 3: Requirement verification and traceability chart (Cameo Systems Modeler).

and runtime signal flow for this configuration is provided in Appendix A.1. The complete test case specifications are provided in Table A.1. Listing A.1 presents the MATLAB test function for TC4.1 as a representative example of the data availability verification approach, and Listing A.2 presents the test function for TC4.6 as a representative example of the update rate verification approach.

Eight automated test functions were implemented in MATLAB, each associated with a requirement through a *Verified By* relationship. The data availability functions (TC4.1-TC4.5) subscribe to the corresponding ROS2 topic published by the Gazebo simulation environment and evaluate whether at least one valid data sample is received within a predefined time window. This binary availability assessment serves as a prerequisite for the update rate tests, as higher-level requirement evaluation depends on the presence of valid sensor data. The update rate functions (TC4.6-TC4.8) collect message timestamps over the 30-second simulation window, compute the average update rate from inter-message intervals, and compare the result against the specified minimum threshold.

All five data availability tests (TC4.1–TC4.5) passed, confirming that the digital replica of the system of interest publishes each required sensor stream and that the MATLAB verification environment can successfully receive and evaluate the data through the ROS2 communication interface. The update rate tests produced mixed results. The IMU update rate (Req 2.3.1) measured 97.4 Hz against a 100 Hz threshold, narrowly failing verification. The LIDAR update rate (Req 2.3.2) measured approximately 2.9 Hz against a 5 Hz threshold, reflecting the sensor's native scan rate in the Gazebo simulation environment. The angle and position update rate (Req 2.3.3) passed verification, exceeding the 20 Hz minimum threshold. These mixed outcomes illustrate distinct engineering scenarios: a near-miss failure that may warrant requirement renegotiation, a significant gap driven by hardware limitations, and successful compliance under a representative constraint.

Beyond the eight primary test cases, a camera data availability requirement was included to demonstrate how the workflow handles requirements that lack

implementation coverage. A test function was executed for this requirement; however, because the TurtleBot3 Burger DTw does not currently include a camera model, no Simulink subsystem component is linked to this requirement, and the test function failed due to the absence of the expected data stream.

Figure 2 illustrates the resulting traceability chart generated using the MATLAB Requirements Toolbox, showing the combined outcomes across all evaluated test cases. In the traceability chart, green and red bars denote passed and failed verification outcomes, respectively, while blue bars indicate requirements allocated to Simulink subsystem components. Unfilled regions indicate requirements lacking architectural allocation, which results in failed verification due to incomplete architectural realization within the system model. In cases where requirements are decomposed into sub-requirements, mixed red and green bars represent aggregated verification outcomes, indicating that some subordinate conditions were satisfied while others were not. Such cases require further analysis during the loop-back update to the MBSE model. As part of the workflow's closed-loop process, all verification results were propagated back to the Cameo model, updating the verification status of each associated requirement. Figure 3 shows the corresponding view in Cameo after the verification status of each requirement was updated, with pass, fail, and partial outcomes reflected directly in the requirement table. The verification and satisfaction matrices in Cameo now reflect these outcomes, establishing a bidirectional traceability chain from requirement definition through DTw-based evaluation to verification status update within the MBSE environment.

5. CONCLUSIONS

This paper presented a DTw–enabled workflow for automated functional requirement verification within a MBSE environment. The workflow establishes a structured DT linking requirement models, system architecture representations, and DTw-based verification activities, enabling automated assessment of requirement compliance while maintaining traceability across engineering artifacts.

The demonstrated implementation using a differential-drive robotic platform confirms the feasibility of integrating DTw into model-based verification processes. The results show how executable system representations can support early verification activities and maintain consistency between design models and verification outcomes. The demonstration produced a mix of passing, failing, and unimplemented verification outcomes, confirming that the workflow can surface actionable engineering findings such as threshold violations, hardware-driven capability gaps, and missing implementation coverage.

Future work will extend the workflow to additional functional verification scenarios, including navigation performance assessment, and will explore its application to more complex cyber-physical systems with larger requirement sets and interdependent verification functions.

References

[1] N. Kass and J. Kolozs, "Getting Started with MBSE in Product Development," *INCOSE International Symposium*, vol. 26, pp. 526–541, July 2016.

[2] M. Hecht and J. Chen, "Verification and Validation of SysML Models," *INCOSE International Symposium*, vol. 31, pp. 599–613, July 2021.

[3] A. Morkevicius and N. Jankevicius, "An approach: SysML-based automated requirements verification," in *2015 IEEE International Symposium on Systems Engineering (ISSE)*, (Rome, Italy), pp. 92–97, IEEE, Sept. 2015.

[4] A. Dagna, S. Centomo, E. Brusa, C. Delprete, and R. Gentile, "AUTOMATIC SYSTEM REQUIREMENTS VERIFICATION FOR THE MBSE-ORIENTED AIRCRAFT DESIGN PROCESS,"

[5] A.-L. Bruggeman, B. Van Manen, T. Van Der Laan, T. Van Den Berg, and G. La Rocca, "An MBSE-Based Requirement Verification Framework to support the MDAO process," in *AIAA AVIATION 2022 Forum*, (Chicago, IL & Virtual), American Institute of Aeronautics and Astronautics, June 2022.

[6] P. Gu, Y. Zhang, Z. Chen, C. Zhao, K. Xie, Z. Wu, and L. Zhang, "X-RMTV: An Integrated Approach for Requirement Modeling, Traceability Management, and Verification in MBSE," *Systems*, vol. 12, p. 443, Oct. 2024.

[7] R. Honcak and A. Wooley, "An MBSE approach for Virtual Verification & Validation of Systems with Digital Twins," in *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, (Linz Austria), pp. 390–400, ACM, Sept. 2024.

[8] I. Bouhali, V. Idasiak, J. Martinez, F. Mhenni, J.-Y. Choley, L. Palladino, and F. Kratz, "A Collaboration Framework Using Digital Twin for Dynamic Simulation and Requirements Verification Based on MBSE and the MIC Concept," in *2024 IEEE International Systems Conference (SysCon)*, (Montreal, QC, Canada), pp. 1–8, IEEE, Apr. 2024.

[9] F. Sahebsara, S. Khatouri, D. Aziz, O. Zeki, M. R. Hieb, and A. K. Raz, "The cornerstone of digital engineering: Harnessing model-based systems engineering for future systems," in *AIAA SCITECH 2026 Forum*, 2026.

[10] S. Khatouri, F. Sahebsara, M. R. Hieb, and A. K. Raz, "From physical realm to digital twin: An additive manufacturing approach to digital twin testbed development," in *AIAA SCITECH 2026 Forum*, 2026.

[11] Object Management Group, "Requirements interchange format (reqif), version 1.2," Specification formal/2016-07-01, Object Management Group, July 2016.

[12] M. Jastram and C. Ebert, " ReqIF: Seamless Requirements Interchange Format between Business Partners ," *IEEE Software*, vol. 29, pp. 82–87, Sept. 2012.

[13] MathWorks, "Import requirements from reqif files." `https://www.mathworks.com/help/slrequirements/ug/import-requirements-from/-reqif-files.html`. Accessed: 2026-03-11.

6. CONTACT INFORMATION

Ali K. Raz
Assistant Professor
George Mason University
College of Engineering and Computing

Systems Engineering and Operations Research
4400 University Drive
Fairfax, Virginia 22030
araz@gmu.edu

7. ACKNOWLEDGMENT

The authors gratefully acknowledge the support from National Center for Manufacturing Sciences (NCMS) for funding this research and enabling advancements in digital engineering and model-based systems development.

We also extend our sincere thanks to the Center for Collision Safety and Analysis (CCSA) at George Mason University for their valuable collaboration, technical guidance, and access to research infrastructure, all of which played a crucial role in the successful execution of this work.

Portions of this manuscript were edited for grammar and clarity using OpenAI's ChatGPT; all conceptualization, design, data collection, analysis, experimentation, interpretation, and conclusions are solely the authors' work.

8. ACRONYMS

CPS	Cyber Physical System
DTw	Digital Twin
DT	Digital Thread
MBSE	Model-Based Systems Engineering
OMG	Object Management Group
SysML	Systems Modeling Language
V&V	Verification and Validation

APPENDIX

Figure A.1 illustrates an example implementation configuration and runtime execution context of the digital twin–enabled verification workflow. The diagram shows the allocation of selected functional requirements to corresponding Simulink subsystem components and their integration with ROS2-based data interfaces used to receive sensor and state information from the Gazebo simulation environment. During runtime execution, the implemented signal pathways support automated verification functions that evaluate requirement compliance based on observed data streams. Representative views of both the simulated digital twin and the physical TurtleBot3 Burger platform are included to provide context for the modeled sensing and navigation capabilities reflected in the verification configuration. This example is intended to illustrate the structural realization and operational data flow of the workflow rather than to provide additional quantitative verification results.

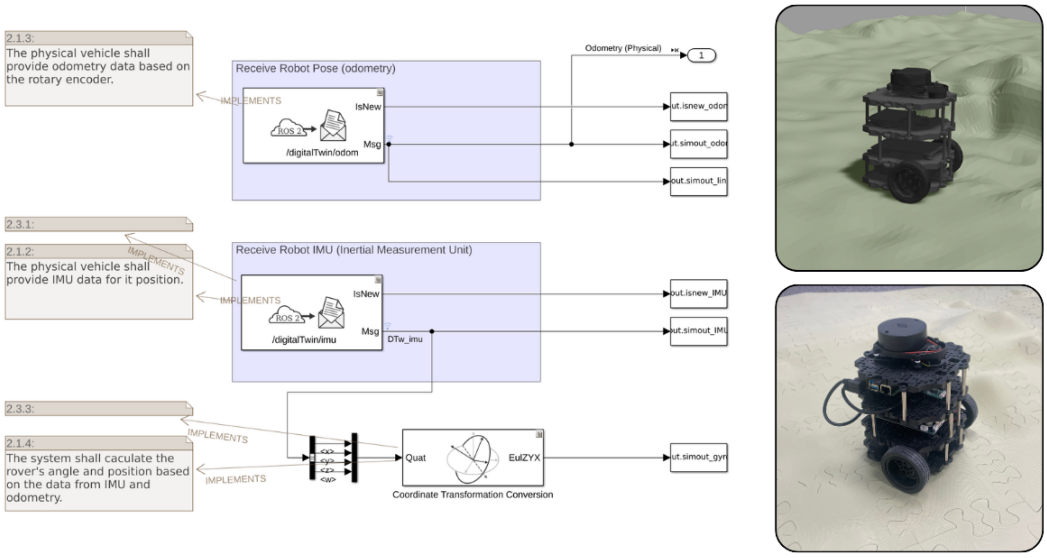

Figure A.1: Example implementation and runtime configuration.

Table A.1: Data availability and update rate test cases executed under test scenario TS2 (digital-only operation). All test cases share the precondition: digital vehicle is running in Gazebo, MATLAB is connected via ROS2, and the vehicle is in an active operation scenario.

ID	Description	Requirement Trace	Comp.	Steps	Expected Result	P/F
TC4.1	IMU values available in MATLAB during operation.	• 2.1.2: The vehicle shall provide IMU data for its position. • 2.2: MATLAB shall store data received from the vehicle for analysis.	IMU Data, MATLAB	Monitor IMU data during operation. Verify continuous updates.	IMU data available in MATLAB during operation.	Pass
TC4.2	Odometry values available in MATLAB during operation.	• 2.1.3: The vehicle shall provide odometry data based on the rotary encoder. • 2.2: MATLAB shall store data received from the vehicle for analysis.	Odometry, MATLAB	Monitor odometry during operation. Verify values change with motion.	Odometry data available in MATLAB throughout motion.	Pass
TC4.3	LIDAR values available in MATLAB during operation.	• 2.1.1: The vehicle shall provide LIDAR data of its surroundings. • 2.2: MATLAB shall store data received from the vehicle for analysis.	LIDAR Data, MATLAB	Monitor LIDAR data during operation. Observe updates.	LIDAR data available in MATLAB during operation.	Pass
TC4.4	Angle (heading/yaw) values available in MATLAB during operation.	• 2.1.4: The system shall calculate the vehicle's angle and position based on IMU and odometry data. • 2.2: MATLAB shall store data received from the vehicle for analysis.	IMU Data, MATLAB	Monitor angle signals during operation. Verify continuous updates.	Angle data stream available in MATLAB throughout operation.	Pass
TC4.5	Position (X/Y) values available in MATLAB during operation.	• 2.1.4: The system shall calculate the vehicle's angle and position based on IMU and odometry data. • 2.2: MATLAB shall store data received from the vehicle for analysis.	Odometry Data, MATLAB	Monitor position signals during operation. Verify continuous updates.	Position data stream available in MATLAB throughout operation.	Pass
TC4.6	IMU data update rate meets minimum threshold during operation.	• 2.3.1: The vehicle shall provide IMU data at a minimum update rate of 100 Hz.	IMU Data, MATLAB	Collect timestamps over 30 s. Compute average rate. Compare against threshold.	IMU update rate $\geq$ 100 Hz.	Fail
TC4.7	LIDAR data update rate meets minimum threshold during operation.	• 2.3.2: The vehicle shall provide LIDAR data at a minimum update rate of 5 Hz.	LIDAR Data, MATLAB	Collect timestamps over 30 s. Compute average rate. Compare against threshold.	LIDAR update rate $\geq$ 5 Hz.	Fail
TC4.8	Computed angle and position data update rate meets minimum threshold during operation.	• 2.3.3: The system shall compute vehicle angle and position at a minimum update rate of 20 Hz.	Odometry Data, IMU Data, MATLAB	Collect timestamps over 30 s. Compute average rate. Compare against threshold.	Angle and position update rate $\geq$ 20 Hz.	Pass

```
classdef TC4_1 < sltest.TestCase
   methods (Test)
      function testIMUReceivedAtLeastOne(testCase)
         model = "scTurtleBot3LogicalArchitecture";
         stopTime = "30";
         simOut = sim(model, "StopTime", stopTime);
         isnew = simOut.get("isnew_IMU");
         received = any(logical(isnew.Data));
         testCase.verifyTrue(received, ...
            "FAIL: No IMU messages were received during the simulation window.");
      end
   end
end
```

Listing A.1: MATLAB test function for TC4.1: IMU data availability verification.

```
classdef TC4_6 < sltest.TestCase
   methods (Test)
      function testIMUUpdateRate(testCase)
         model = "scTurtleBot3LogicalArchitecture";
         stopTime = "30";
         minRate = 100; %Hz
         simOut = sim(model, "StopTime", stopTime);
         isnew = simOut.get("isnew_IMU");
         newMsgTimes = isnew.Time(logical(isnew.Data));
         dt = diff(newMsgTimes);
         avgRate = 1 / mean(dt);
         testCase.verifyGreaterThanOrEqual(avgRate, minRate, ...
            sprintf("FAIL: IMU update rate %.2f Hz is below %.0f Hz threshold.", ...
            avgRate, minRate));
      end
   end
end
```

Listing A.2: MATLAB test function for TC4.6: IMU update rate verification.