LIGHTHOUSE

SWEDISH MARITIME COMPETENCE CENTRE

LIGHTHOUSE REPORTS

Virtual Wind Ship

An innovation project carried out within the Swedish Transport Administration's industry program Sustainable Shipping, operated by Lighthouse, published February 2026

www.lighthouse.nu

LIGHTHOUSE
SWEDISH MARITIME COMPETENCE CENTRE

HÅLLBAR SJÖFART

Virtual Wind Ship

A multi-stakeholder test bench

Authors

Jan Östh from RISE Research Institute of Sweden

Luis Sanchez-Heres from RISE Research Institute of Sweden

Fredrik Olsson from RISE Research Institute of Sweden

In cooperation with

Carl Fagergren from Wallenius Marine

Fredrik Roos from AlfaWall Oceanbird

Ulysse Dhome from KTH Royal Institute of Technology

Daniel Koch from Manta Marine

Summary

The Virtual Wind Ship project developed a distributed co-simulation platform that enables multiple organizations to collaboratively simulate complex interconnected ship systems while protecting their intellectual property. The platform addresses a key barrier to wind propulsion adoption: the need for holistic system simulations that span organizational boundaries, where no single party possesses all the required models and knowledge.

The project produced Liaison, an open-source tool that combines the FMI 3.0 simulation standard with the Zenoh communication protocol. It is released under the Apache 2.0 license and is available at https://github.com/RISE-Maritime/liaison. Liaison allows simulation models to remain within each organization's infrastructure while participating in shared simulations over the Internet. The tool generates lightweight client FMUs that contain only interface descriptions and communication logic, enabling collaboration without transferring proprietary model implementations.

The platform was validated through co-simulations of a generic wind-assisted cargo vessel, executed across three geographically distributed sites connected via the internet. The demonstrations successfully quantified fuel savings from wind-assisted propulsion, and additional savings when an intelligent propulsion control system was engaged, for the selected scenarios that were simulated. Furthermore, the test also identified deficiencies in the provided sail control system, illustrating how the platform enables early detection of integration issues before physical installation, providing valuable feedback early during the design phase of the system.

Sammanfattning

Projektet Virtual Wind Ship har utvecklat en distribuerad plattform för co-simulering som gör det möjligt för flera organisationer att gemensamt simulera sammanlänkade komplexa fartygssystem samtidigt som deras immateriella rättigheter skyddas. Plattformen adresserar ett centralt hinder för införandet av vinddrift: behovet av helhetssimulering av system som spänner över organisationsgränser, där ingen enskild part besitter alla nödvändiga modeller och kunskaper.

Projektet har resulterat i Liaison, ett verktyg med öppen källkod som kombinerar simuleringsstandarden FMI 3.0 med kommunikationsprotokollet Zenoh. Verktyget har en Apache 2.0 licens och finns tillgängligt på https://github.com/RISE-Maritime/liaison. Liaison gör det möjligt för simuleringsmodeller att förbli inom respektive organisations infrastruktur samtidigt som de deltar i gemensamma simuleringar över internet. Verktyget genererar lättviktiga klient-FMU:er som endast innehåller gränssnittsbeskrivningar och kommunikationslogik, vilket möjliggör samarbete utan att överföra proprietära modellimplementationer.

Plattformen validerades genom sam-simuleringar av ett generiskt vindassisterat lastfartyg, utförda över tre geografiskt distribuerade platser anslutna via internet. Demonstrationerna kvantifierade bränslebesparingar från vindassisterad framdrivning och ytterligare besparingar när det intelligent propulsionskontrollsystemet var påkopplat. Testerna identifierade även brister i segelstyrningssystemet, vilket illustrerar hur plattformen möjliggör tidig upptäckt av integrationsproblem före fysisk installation och tillhandahåller värdeful återkoppling tidigt under utvecklingsfasen av systemet.

Contents

Abbreviations

CS	Co-Simulation
DCP	Distributed Co-Simulation Protocol
FMI	Functional Mockup Interface
HIL	Hardware-in-the-loop
IP	Intellectual Property
JVM	Java Virtual Machine
ME	Model Exchange
NTNU	Norwegian University of Science and Technology
OSP	Open Simulation Platform
SIL	Software-in-the-loop
VWS	Virtual Wind Ship (Name of project)
WAPS	Wind Assisted Propulsion Systems
SIL	Software-in-the-loop

1 Introduction

This report presents the outcomes of the "Virtual Wind Ship" (VWS) project, an innovation project carried out within the Swedish Transport Administration's industry program Sustainable Shipping, managed by Lighthouse. The project addresses the challenge of enabling collaborative system simulations across multiple organizations while protecting intellectual property.

1.1 Background

Modern ships are complex technological systems comprising not only structural components such as hulls, propellers, rudders, and superstructures, but also advanced integrated systems combining mechanical, electrical, and software elements. These integrated systems rely heavily on software for control and coordination: engine control units regulate fuel injection and emissions, autopilot systems manage course keeping, and energy management systems optimize power distribution across the vessel. The propulsion and steering systems exemplify this integration, combining physical components (propellers, shafts, rudders, hydraulic actuators) with sophisticated control algorithms that must work together seamlessly.

These systems are critical for vessel operation, and significant engineering effort is invested in their design and integration. The interaction between propulsion and steering systems, combined with hydrodynamic and environmental effects from ocean waves and wind forces, determines the ship's maneuverability characteristics that define whether a vessel can safely operate in its intended conditions.

The introduction of Wind Assisted Propulsion Systems (WASPs), such as rotor sails or wing sails, adds another layer of complexity. A WASP directly interacts with existing propulsion and steering systems as well as the hydrodynamic and aerodynamic forces acting on the hull.

Understanding how a WASP affects ship performance requires system simulations that can evaluate these interactions during the design phase or when retrofitting existing vessels. To validate that control systems work correctly, engineers can also use Software-in-the-loop (SIL) and Hardware-in-the-loop (HIL) testing. In SIL testing, control algorithms run in a simulated environment alongside models of the ship. HIL testing goes further by connecting actual hardware (such as control units or sensors) to the simulation. These testing approaches are well established in the automotive and aerospace industries but remain underutilized in the maritime sector.

However, performing such holistic simulations presents organizational challenges. Typically, no single organization possesses all the knowledge and models required. Propulsion system suppliers, steering system manufacturers, ship designers, and independent evaluation bodies each work with their own simulation tools and models.

Building accurate models of components supplied by others requires detailed engineering knowledge that suppliers are often unwilling to share, as it constitutes their intellectual property (IP).

A promising approach to address these challenges is distributed co-simulation, where each organization retains their models within their own infrastructure while connecting them remotely to a shared simulation environment. This preserves IP protection, simplifies version control, and enables more fluid collaboration across organizational boundaries.

1.2 Purpose

The purpose of this project is to develop a distributed co-simulation platform that enables Software-in-the-loop (SIL) and Hardware-in-the-loop (HIL) co-testing of control systems for wind-powered vessels. The platform will allow multiple organizations to collaboratively build virtual ship models with all its subsystems during the design phase, reducing time-to-market for wind propulsion technology by enabling testing against these virtual representations before physical installation.

Each organization retains their simulation models within their own infrastructure while connecting them remotely to a shared simulation environment. This approach protects intellectual property and removes barriers to collaboration across organizational boundaries.

1.3 Objectives

The project has the following objectives:

1. **Develop a distributed co-simulation platform** for Software-in-the-loop (SIL) and Hardware-in-the-loop (HIL) co-testing of different actors' control systems and hardware for wind-powered vessels.
2. **Enable virtual ship models** to be collaboratively built during the design phase, allowing systems to be tested before physical installation.
3. **Protect intellectual property** by designing the platform so that simulation models remain within each organization's infrastructure, with communication flowing only through a standardized API.
4. **Reduce time-to-market** for wind propulsion technology by allowing suppliers to develop and calibrate their control systems against a virtual ship representation before installation on real vessels.
5. **Support the transition to sustainable shipping** by facilitating the development of primary wind-powered ships that use wind as the main means of propulsion.

1.4 Participants

The project partners are:

- RISE Research Institutes of Sweden
- KTH Royal Institute of Technology
- Alfawall Oceanbird
- Wallenius Marine
- Manta Marine (formerly Yara Marine Technologies)

1.5 Structure of the report

The report is organized as follows:

- **Requirements and design** presents the system requirements specification and reviews existing open-source software and standards relevant to the project.
- **Implementation** describes the platform architecture and the Liaison software developed in the project.
- **Demonstration** presents test cases and results from platform validation.
- **Conclusions** summarizes the project outcomes.
- **Future work** outlines potential directions for continued development.

2 Requirements and design

2.1 Specification of system requirements

The following requirements were identified as high-level strict technical requirements in order to fullfill the objectives outlined in 1.3:

1. **Enable co-simulation without IP transfer**: Organizations must be able to participate in collaborative simulations without transferring their intellectual property to other parties.
2. **Use existing simulation interfaces**: The platform must use established standards to maximize interoperability with existing simulation tools.
3. **Minimal configuration and ease of use**: The platform should minimize the need for complex network configuration, such as opening ports on host computers or managing IP addresses directly. Additionaly, the platform should be straightforward to use.

2.2 Review of existing open-source software and standards

This section reviews the relevant standards and open-source software that are relevant for this project.

2.2.1 Standards

Functional Mock-up Interface (FMI)

The Functional Mock-up Interface (FMI) [1] is a free standard that defines a container and an interface to exchange dynamic simulation models using a combination of XML files, binaries and C code, distributed as a ZIP file. FMI defines a set of conventions and interfaces that govern how simulation models interact with simulation environments, allowing for the encapsulation of models as Functional Mock-up Units (FMUs). These FMUs encapsulate simulation models along with their associated parameters, inputs, and outputs, enabling their exchange and utilization across diverse simulation workflows.

Concretely, an FMU is a zip package (with a ".fmu" file extension) that contains XML files with information on inputs, outputs, and connections, as well as compiled C code containing the original simulation model wrapped in a standardized interface layer.

The previous versions, FMI 1.0 and 2.0, have two interface types:

- **Model Exchange (ME)**: Exposes an ordinary differential equation (ODE) to an external solver. The importer's integration algorithm controls time advancement, state management, and event handling. This approach offers maximum flexibility

but requires the importer to implement numerical integration. The FMU contains only the model equations.

- **Co-Simulation (CS)**: Packages models with their own solvers or schedulers. Time advances through negotiated communication steps between the importer and FMU, enabling coupling of heterogeneous simulation tools. This reduces importer complexity compared to Model Exchange and includes features such as early return for event handling, Event Mode, and Intermediate Update Mode.

The latest version, FMI 3.0, introduced a third interface type:

- **Scheduled Execution (SE)**: An interface type that exposes individual model partitions for external scheduler control. It is specifically designed to support virtual electronic control units and real-time execution environments. The scheduler controls partition execution timing and supports clocks for precise event synchronization, enabling model partition preemption.

FMI is backed by over 250 tools, making it the de-facto industry standard for simulation model exchange. The standard is maintained as a Modelica Association Project with an Advisory Committee including major industry players such as Siemens, Bosch, Boeing, and Airbus.

A notable companion standard is the System Structure and Parametrization (SSP) [2], which defines a tool-independent format for the description, packaging, and exchange of complete systems consisting of one or more FMUs.

Distributed Co-Simulation Protocol (DCP)

The Distributed Co-Simulation Protocol (DCP) [3] is a communication protocol that enables different simulation systems to work together over a network. Developed through the ACOSAR project and maintained as a Modelica Association Project, DCP uses a coordinator-participant architecture: a central DCP master orchestrates the simulation while DCP participants (called slaves) perform the actual computations. These participants can be software simulations or physical hardware, making DCP suitable for both pure virtual testing and hardware-in-the-loop scenarios.

A DCP simulation progresses through five phases: registration (participants announce themselves), configuration (the master sets up communication paths), initialization (bringing all participants to a consistent starting state), synchronization (ensuring all participants are ready), and running (the actual simulation). The protocol supports different timing modes, from simulations that run as fast as possible to strict real-time execution where simulated time must match wall clock time exactly.

DCP was designed with FMI compatibility in mind. While FMI provides a standard for packaging and exchanging simulation models, it lacks native support for distributed execution. DCP complements FMI by providing the communication layer needed to execute FMI-based models across distributed systems. Data types can be translated

between the two standards, allowing FMUs to be wrapped as DCP participants and integrated into distributed co-simulation scenarios.

2.2.2 Open-source platforms and tools

OSP - Open Simulation Platform

The Open Simulation Platform (OSP) [4, 5] was developed by a consortium initiated by DNV GL, Kongsberg Maritime, SINTEF, and Norwegian University of Science and Technology (NTNU)to address the need for co-simulations with a common nomenclature and methods tailored for the maritime sector. OSP was motivated by the increasing complexity of modern ship systems, which have become increasingly software-driven and integrated. The platform addresses challenges in predicting integrated system behavior, assessing system safety across numerous components by different vendors, and managing updates to individual software components in multi-vendor environments.

OSP comprises four primary elements:

- **libcosim**: A C/C++ library for co-simulation that enables integration into various simulation software. The library includes bindings for Python, Java, and C.
- **OSP Interface Specification (OSP-IS)**: A specification that extends FMI 2.0 by adding semantic meaning to model interface variables, simplifying the model connection process and enabling validation of semantically sound simulations.
- **OSP Reference Models**: A collection of maritime reference models representing typical maritime equipment, provided as example FMUs and system configurations.
- **Demo applications and tools**: Reference implementations including a command-line interface and demonstration applications.

OSP uses the FMI standard as its technical foundation. Rather than replacing FMI, OSP-IS adds a semantic layer on top of FMI 2.0. While FMI handles the mechanical aspects of connecting models (data types, causality, variability), OSP-IS introduces an ontology that defines what model variables represent and how they can be connected.

The key contribution of OSP-IS is the concept of variable groups. Variable groups bundle multiple related FMI variables into single connection units, enabling connections to be defined at a higher level rather than requiring scalar variable-by-variable connections. Each variable group has a defined type (such as Force, Torque, Voltage, or LinearVelocity), and groups can only be connected to other groups of compatible types. This approach reduces configuration complexity, enables automatic unit conversions, and allows validation of the system configuration before simulation.

Comparison with FMI 3.0 terminals

FMI 3.0 introduced a similar concept called terminals, which group variables semantically to ease connecting compatible signals between FMUs. Terminals are defined in an optional file within the FMU package and can include graphical representations showing their positioning and appearance.

Both OSP variable groups and FMI 3.0 terminals aim to solve the same problem: simplifying connections between FMUs by grouping related variables and providing semantic information about their intended use. The OSP approach predates FMI 3.0 terminals and was developed specifically for maritime applications. With FMI 3.0 now available, the terminal concept has become part of the standard itself, potentially reducing the need for domain-specific extensions in future implementations.

FMU-proxy

FMU-proxy [6, 7, 8] is an open-source framework developed at NTNU that enables remote access to FMUs over a network. It works by wrapping an FMU inside a server application that exposes the FMU's interface through Remote Procedure Calls (RPC). Client applications can then interact with the FMU as if it were running locally, even though the actual computations occur on a remote machine. This approach allows FMUs to be accessed from different programming languages and platforms without requiring native FMI library support on the client side.

FMU-proxy consists of several components:

- **fmu-proxify**: A command-line tool that transforms a co-simulation FMU into a proxified version with network capabilities. The proxy FMU behaves like a standard FMU externally while communicating with the original FMU over the network.
- **fmu-proxy-server**: A server application that wraps the original FMU. Each new instance of the original FMU runs in a separate process, which also enables running multiple instances of FMUs that normally only allow one instance per process.
- **Client libraries**: RPC client implementations for C++, Python, JavaScript, and the Java Virtual Machine (JVM), enabling access to the original FMU from multiple programming environments without requiring native FMI library implementations.

Server implementations exist in both C++ and for the JVM. The C++ implementation uses FMI Library for handling FMUs, while the JVM version uses FMI4j.

FMU-proxy enables distributed co-simulation scenarios by allowing FMUs to run on remote resources. The framework addresses several practical challenges:

1. **Cross-version compatibility**: Import FMI 1.0 models into software that only supports FMI 2.0.

2. **Multi-instance support**: Run multiple instances of FMUs that otherwise only allow one instance per process.
3. **Remote execution**: Execute FMUs on remote machines, allowing FMUs to run on platforms that would otherwise be unsupported by the simulation environment.
4. **Language independence**: Access FMUs from any programming language that supports the chosen RPC technology, without requiring native FMI library implementations.

Zenoh

Zenoh [10] is a communication protocol that operates on named resources, structured into an application-level key-space, rather than directly with hardware-bound addresses such as IP addresses and port numbers. The protocol can function on top of a TCP/IP stack but also on other communication stacks such as Bluetooth or even across a serial line.

This distinction allows for flexible system infrastructures supporting peer-to-peer communication, routed communication, and brokered communication. This manifests as the possibility to create cliques of separate simulation nodes, several interconnected cliques, or a fully distributed set of simulation nodes interconnected through a web of communication points (brokers).

The protocol supports both messaging using the publish-subscribe paradigm as well as the request-response paradigm, making it a flexible choice for implementing various software architectures.

2.3 Gap analysis

The following analysis evaluates each reviewed technology against the three system requirements.

2.3.1 FMI (Functional Mock-up Interface)

Requirement	Assessment
1. Co-simulation without IP transfer	**Partial.** FMI defines a standard interface for co-simulation but does not address distributed execution. FMUs must be transferred to the simulation environment, which requires sharing compiled binaries containing model logic.
2. Use existing simulation interfaces	**Yes.** FMI is the de-facto industry standard. FMI 3.0 introduces improvements relevant to this project, including Scheduled Execution and Terminals.

Requirement	Assessment
3. Minimal configuration and ease of use	**Not applicable.** FMI is an interface standard, not a deployment solution. Configuration and usability depend on the tools used to execute FMUs.

2.3.2 DCP (Distributed Co-Simulation Protocol)

Requirement	Assessment
1. Co-simulation without IP transfer	**Yes.** DCP was designed for distributed execution, allowing simulations to run on separate machines while coordinating through a central master. Models can remain within their owner's infrastructure.
2. Use existing simulation interfaces	**Partial.** DCP is compatible with FMI but is a separate standard. Integration requires additional development effort.
3. Minimal configuration and ease of use	**No.** DCP relies on traditional network protocols requiring port configuration and IP address management. Deployment across organizational boundaries requires coordination with IT departments. The standard is not as widely adopted as FMI, resulting in fewer available tools and resources. Setting up a DCP-based simulation requires significant expertise.

2.3.3 OSP (Open Simulation Platform)

Requirement	Assessment
1. Co-simulation without IP transfer	**No.** OSP focuses on co-simulation within a single environment. All FMUs must be present on the same system, requiring transfer of model files.
2. Use existing simulation interfaces	**Partial.** OSP builds on FMI but only supports versions 1.0 and 2.0, missing improvements introduced in FMI 3.0 such as Scheduled Execution and Terminals.
3. Minimal configuration and ease of use	**Partial.** OSP simplifies model connections through variable groups and semantic typing, making local co-simulations straightforward to set up. However, it does not address network configuration for distributed scenarios.

2.3.4 FMU-proxy

Requirement	Assessment
1. Co-simulation without IP transfer	**Yes.** FMU-proxy enables remote FMU access over a network. The original FMU can remain on the owner's infrastructure while clients interact with it remotely.
2. Use existing simulation interfaces	**Partial.** FMU-proxy supports FMI 1.0 and 2.0 but not FMI 3.0. This excludes features such as Scheduled Execution and Terminals.
3. Minimal configuration and ease of use	**No.** FMU-proxy uses RPC technologies (Thrift, gRPC, JSON-RPC) that require opening specific ports and managing IP addresses. The project has not seen active development in five years, raising concerns about maintainability and availability of documentation and support.

2.3.5 Zenoh

Requirement	Assessment
1. Co-simulation without IP transfer	**Not applicable.** Zenoh is a communication protocol, not a simulation framework. It provides the transport layer but requires additional software to enable co-simulation.
2. Use existing simulation interfaces	**No.** Zenoh does not implement FMI or other simulation interfaces. Integration with FMI-based workflows requires custom development.
3. Minimal configuration and ease of use	**Yes.** Zenoh operates on named resources in an application-level key-space rather than hardware-bound addresses. Using a router deployed on a well-known address, nodes can connect without opening ports or managing IP addresses directly. The protocol is well-documented and actively maintained with bindings for multiple languages.

2.3.6 Summary

The following table summarizes the gap analysis:

Technology	Req. 1: IP Protection	Req. 2: FMI Support	Req. 3: Config & Ease of Use
FMI	Partial	Yes	N/A
DCP	Yes	Partial	No
OSP	No	Partial	Partial.
FMU-proxy	Yes	Partial	No
Zenoh	N/A	No	Yes

No single existing solution satisfies all three requirements. However, combining FMI 3.0 (for standardized simulation interfaces) with Zenoh (for minimal configuration and ease of use) addresses the identified gaps. This combination forms the basis for the Liaison tool developed in this project.

3 Implementation

This section describes the distributed co-simulation platform developed in the project. We first introduce Liaison, the open-source tool developed in the project, followed by a detailed description of the platform architecture.

3.1 Liaison

Liaison is an open-source tool for distributed co-simulations based on the FMI 3.0 standard and the Zenoh communication protocol. It is released under the Apache 2.0 license and is available at https://github.com/RISE-Maritime/liaison.

3.1.1 Technology stack

Liaison is implemented in C++ and compiled for Windows and Linux. While it depends on standard system libraries, third-party dependencies are bundled with the release, requiring no additional installation.

The choice of technology stack reflects the project requirements:

- **FMI 3.0**: The latest version of the de-facto industry standard for co-simulation.
- **Zenoh**: A pub/sub/query protocol that supports flexible network topologies with minimal configuration.
- **C++**: Provides the performance characteristics required for real-time simulation scenarios and compatibility with the FMI standard's C-based interface.

3.1.2 Liaison components

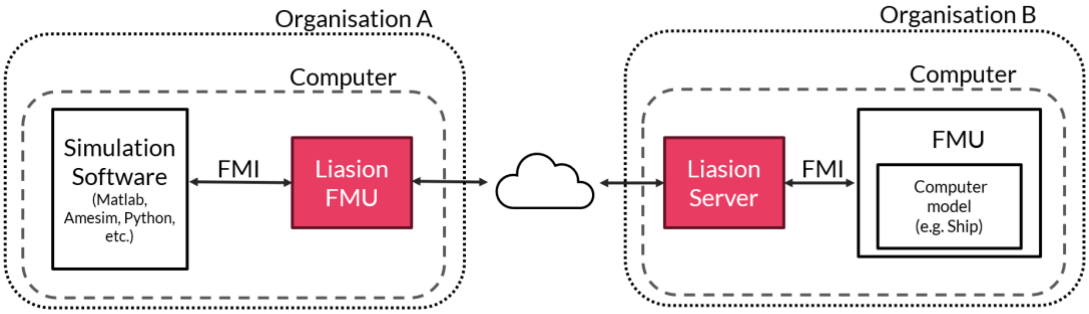
Liaison consists of a platform-specific executable and a set of dynamic libraries that encapsulate the client logic for different platforms. The executable performs two main functions:

1. **Creating a Liaison FMU**: Given a source FMU, the tool extracts the model description XML file, renames the dynamic libraries according to the model's name, bundles the client communication logic, and re-packages everything into a new FMU. The resulting Liaison FMU contains all information needed to communicate with a Liaison Server, including network configuration.
2. **Serving an FMU**: The executable loads a source FMU and makes it available over the network. It listens for incoming requests from Liaison FMUs and forwards all function calls to the loaded FMU, returning the results to the caller.

3.1.3 Workflow

A typical workflow for using Liaison involves the following steps:

1. **Model owner creates Liaison FMU**: Starting with an existing FMU, the model owner uses Liaison to create a Liaison FMU (the client) while retaining the original FMU for serving.
2. **Distribution of Liaison FMU**: The model owner sends the Liaison FMU to the simulation operator. This Liaison FMU contains no intellectual property from the original model, only the interface description and communication logic.
3. **Server deployment**: The model owner deploys the Liaison Server with the original FMU within their own infrastructure.
4. **Simulation execution**: The simulation operator loads the Liaison FMU into their simulation software. When the simulation runs, the Liaison FMU communicates with the Liaison Server to execute the actual model computations. From the simulation software's perspective, the Liaison FMU behaves like any standard FMU.



3.1.4 Current implementation status

At the time of writing, Liaison implements a subset of the FMI 3.0 standard sufficient for co-simulation scenarios. The implemented functionality covers the core operations required for stepping simulations forward and exchanging data between components. For an up-to-date list of implemented functions, refer to the project's online repository.

3.2 Architecture

The platform follows a server-client architecture where simulation models remain within the control of their owners while being accessible to remote simulation environments. The architecture involves four types of components working together:

- **Co-simulation engine**: The simulation software responsible for orchestrating the co-simulation. The co-simulation engine loads one or more FMUs, manages

the connections between them, and steps the simulation forward in time. The co-simulation engine initiates all simulation activities and controls the simulation timeline. Examples of co-simulation engines include Ecos, MATLAB Simulink, and OpenModelica.

- **Original FMU**: The source FMU containing the actual simulation model. This is a standard FMU created by the model owner using their preferred simulation tool, such as MATLAB Simulink, Siemens Simcenter Amesim, and OpenModelica. The original FMU encapsulates the model's intellectual property and remains within the model owner's infrastructure. It is never transferred to other parties.
- **Liaison Server**: A server application that loads and hosts an original FMU. The server receives incoming function calls from clients over the network and forwards them to the original FMU, returning the results. The Liaison Server must be running before a simulation can begin, as it provides access to the actual computational model.
- **Liaison FMU**: A client application packaged as an FMU that substitutes the original FMU in the co-simulation engine. The Liaison FMU is generated from the original FMU but contains no intellectual property, only the interface description (inputs, outputs, parameters) and communication logic. From the perspective of the co-simulation engine, the Liaison FMU appears and behaves as a standard FMU. Internally, it translates all FMI function calls into network requests to a Liaison Server.

In a distributed co-simulation setup, the co-simulation engine remains the central orchestrator. The engine loads Liaison FMUs just as it would load regular FMUs. When the engine calls FMI functions on a Liaison FMU (such as setting input values, getting output values, or advancing simulation time), the Liaison FMU forwards these calls over the network to its corresponding Liaison Server. The Liaison Server executes the function on the original FMU and returns the result.

This design means that the co-simulation engine does not need any modifications to work with distributed models. From its perspective, all FMUs are local. The distribution is transparent to the simulation software.

The simulation flow proceeds as follows:

1. Liaison Servers are started on the machines hosting the original FMUs
2. The co-simulation engine loads the Liaison FMUs
3. The co-simulation engine initializes all FMUs (Liaison FMUs forward initialization calls to their servers)

4. The co-simulation engine steps the simulation forward, exchanging data between FMUs at each communication step
5. When the simulation completes, the co-simulation engine terminates all FMUs

3.2.1 Multi-model configurations

A co-simulation typically involves multiple FMUs representing different subsystems. With Liaison, several configurations are possible:

- **All models distributed**: Every FMU in the simulation is served remotely via Liaison. Each model owner runs their own Liaison Server, and the simulation operator's co-simulation engine connects to all of them through Liaison FMUs.
- **Mixed local and distributed**: Some FMUs are loaded directly by the co-simulation engine while others are accessed via Liaison. This is useful when some models can be shared directly while others require IP protection.
- **Multiple models from the same organization**: A single organization may serve multiple FMUs, each through its own Liaison Server instance. Alternatively, future versions may support serving multiple FMUs from a single server.

3.2.2 Deployment configurations

The architecture supports three deployment configurations, each addressing different use cases:

1. **Local deployment**: Both the Liaison FMU and Liaison Server run on the same physical computer. This configuration is useful for addressing security concerns (running the FMU in a sandbox or Docker environment) or portability issues (running the FMU with specific dependencies isolated from the main system).
2. **LAN deployment**: The Liaison FMU and Liaison Server run on different computers within the same local area network. This enables using dedicated hardware resources or specific operating systems for the FMU execution while running the co-simulation engine elsewhere.
3. **WAN deployment**: The Liaison FMU and Liaison Server run on computers in different networks, connected via the internet. This configuration enables distributed co-simulations across organizations, where each party retains their models within their own infrastructure. The original FMU never leaves the owner's premises, protecting intellectual property while enabling collaborative simulations. This configuration avoids the need for non-disclosure agreements, software licenses, or obfuscation when collaborating on simulations.

The following table summarizes the issues addressed by each deployment configuration:

Issue	Local	LAN	WAN
FMU portability	Yes	Yes	Yes
Security (sandboxing)	Yes	Yes	Yes
IP protection	No	No	Yes

3.2.3 Network topology

The platform uses Zenoh as its communication layer, which supports multiple network topologies:

Peer-to-peer mode (default): Every node discovers other nodes through a discovery mechanism and maintains direct connections to all discovered nodes. This mode enables seamless, zero-configuration connections within a LAN and is the default operating mode.

Client mode: Every node connects to a well-known router on startup. The router ensures end-to-end connectivity between all connected nodes. This mode is particularly useful for WAN deployments where nodes in different LANs need to interconnect. Each node only needs to know the address of the router, not the addresses of other nodes. A router deployed on a well-known address accessible over the internet enables geographically distributed nodes to connect without complex network configuration such as opening ports on host computers.

The underlying Zenoh library can be configured for secured communication channels using either client/server TLS or mutual TLS (mTLS), regardless of network topology.

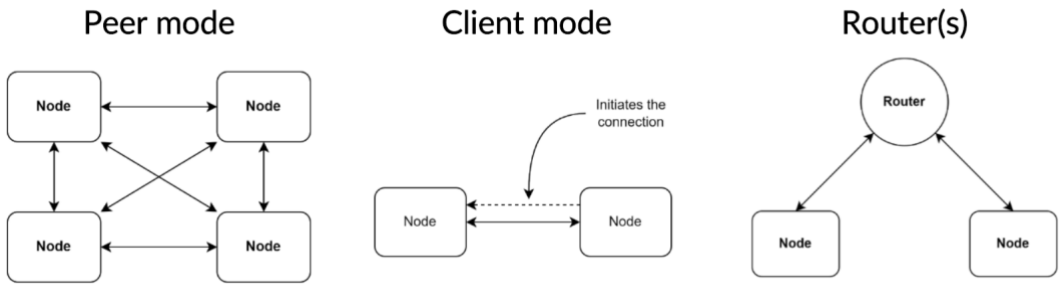


Figure 1: Communication types of Zenoh

4 Demonstration

4.1 Test cases

The distributed co-simulation platform was demonstrated through a simulated test cases involving a wind-assisted cargo vessel. The simulations integrated four FMU components representing the ship dynamics, sail aerodynamics, sail control, and propulsion control subsystems.

4.1.1 Simulation components

Cargo Ship FMU: A hydrodynamic model built using the RISE in-house ship dynamics solver SEAMAN, simulating vessel maneuvering behavior including responses to propulsion forces, rudder commands, and environmental loads. The main characteristics of the cargo vessel are summarized below.

Parameter	Value	Parameter	Value
Length over all	90.3 m	No. of propellers	1
Length between perpendiculars	88.3 m	Propeller type	Fix pitch
Beam	15.5 m	Propeller diameter	3.0 m
Displacement	4420 m^3	No. of engines	1
Aft draught	4.2 m	Engine type	Diesel
Forward draught	4.2 m	Power at MCR	2000.0 kW
GM	0.93 m	Design speed at MCR	8.0 knots
No. of rudders	1	No. of wing sails	1
Rudder max angle	35 deg	Sail area	~400 m^2

- **Sail Wing FMU**: Models the aerodynamic behavior and actuator dynamics of a rigid wing sail using tabulated coefficients that vary by flap angle and angle of attack.

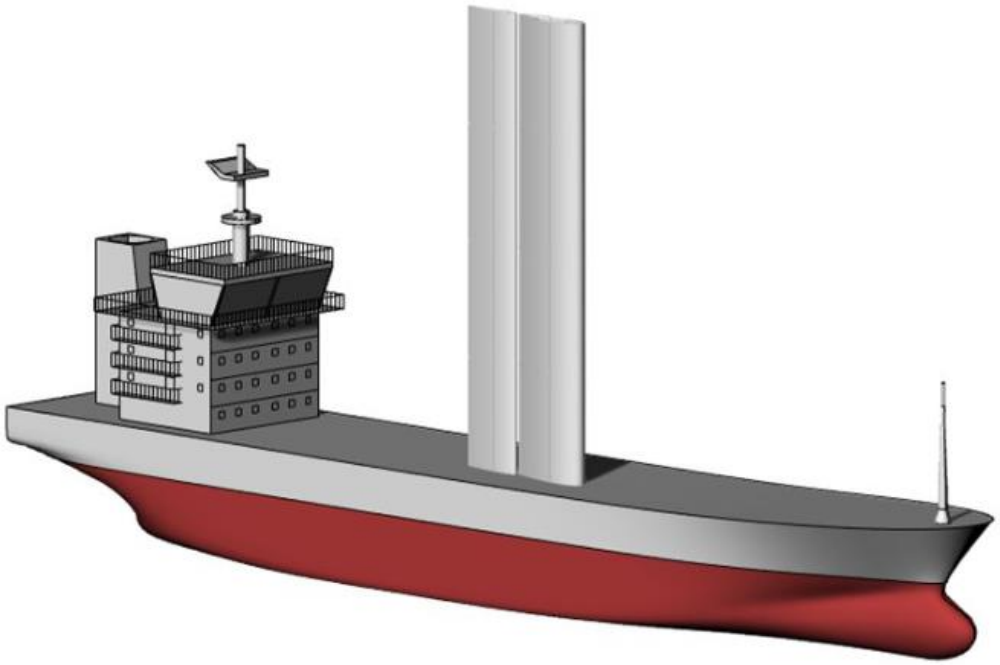


Figure 2 Cargo ship with sail wing.

- **Sail Control System FMU**: Implements proportional control logic for sail positioning, ordering the boom and flap angles of the sail to achieve a specified angle of attack.
- **Propulsion Control System FMU**: Implements fuel consumption control logic for coordinating engine power with sail-generated thrust, through ordering desired rpm values for the engine.

4.1.2 Co-simulation setup

The co-simulations were orchestrated using the Ecos software [9] and executed across three geographically distributed locations in the Gothenburg area:

- Site 1: Sail Wing FMU
- Site 2: Propulsion Control System FMU and Sail Control System FMU
- Site 3: Cargo Ship FMU and Ecos orchestration software

Each site ran a Liaison server exposing its local FMU interfaces, with all nodes connected via the Internet through a Zenoh router. This configuration demonstrates the WAN deployment mode described in the Implementation section, where each organization retains their models within their own infrastructure.

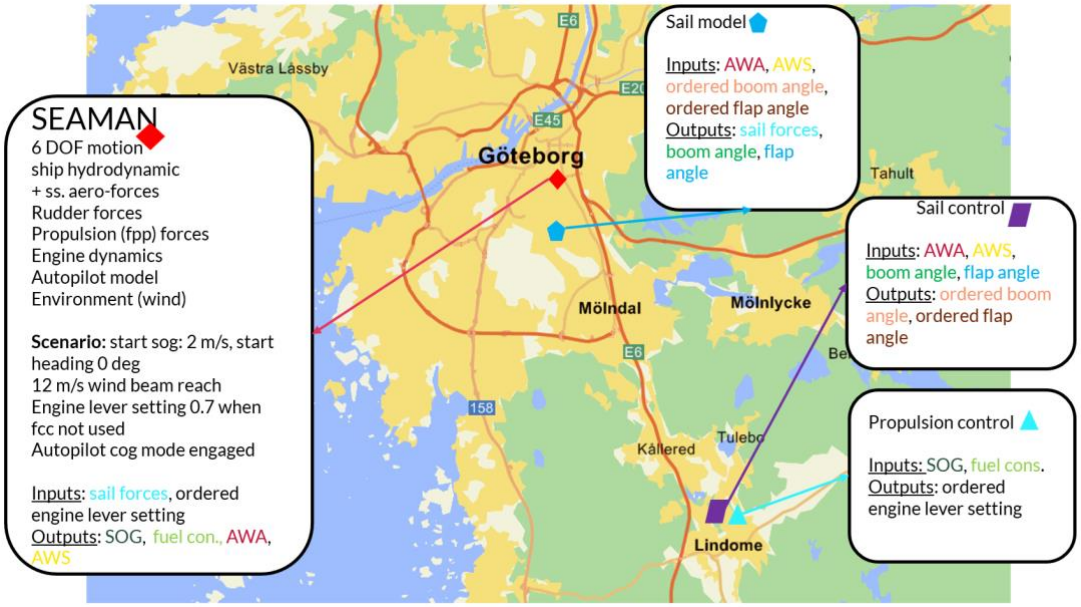


Figure 3 Distributed co-simulation setup.

4.1.3 Test case descriptions

Base scenario

All the following test cases are based on the start scenario described here. Initially, at time = 0, the ship starts with a speed over ground of ~4 knots (2 m/s) and a heading and course over ground of 0° north. The wind speed is 12 m/s and is coming from the east, at a beam reach on the ship. The ship model has an inbuilt autopilot system that is engaged in course over ground mode with a target course of 0° north. In the cases where the propulsion control system is not engaged, the propulsion is controlled manually by a lever setting at 70% of its maximum engagement. The lever setting will in turn set the engine rate of revolution to the value corresponding to 70% of it's maximum engine rate according to the its combinator settings.

Test A: Sail Contribution to Propulsion

This test compares the fuel consumption when the vessel operates with and without sails engaged. The propulsion control system is not engaged, but the propulsion is manually controlled by the lever as described in the base scenario. The comparison quantifies the fuel savings attributable to wind-assisted propulsion under the simulated wind conditions.

Test B: Fuel Consumption Control Effectiveness

This test evaluates the propulsion control system's fuel-saving capabilities. With sails engaged throughout, two configurations are compared: without and with the propulsion

control system engaged. The comparison demonstrates the additional fuel savings achieved when the propulsion controller reduces engine power in response to sail contributions to the total thrust exerted on the ship.

Test C: Sail Control Response to Wind Changes

This test verifies the sail control system's response to changing wind conditions. The vessel sails on a steady course while the wind direction changes 180 degrees. The test monitors the sail boom angle, flap angle, and resulting track to validate that the sail controller correctly repositions the sails to provide maximum forward thrust.

Test D: Sail Control Response to Course Changes

This test verifies the sail control system's response to vessel heading changes. While sailing under constant wind conditions, the vessel executes a 90-degree course change to port. The test monitors the sail angles and forces as the controller adjusts to the new apparent wind angle.

4.1.4 Results and discussion

Test A: Sail Contribution to Propulsion

The results in Fig. 4 and Fig. 5 show that engaging the sails provides a measurable reduction in fuel consumption while maintaining comparable voyage progress. Both configurations followed a northward course over the 2000-second simulation, with the sail-assisted vessel exhibiting a slight lateral deviation due to the sudden influence of the sail forces at the beginning of the simulation.

Both configurations started with similar consumption rates during the initial acceleration phase (0-250 seconds), after which the curves diverged. By the end of the simulation:

- Baseline configuration (no sails): approximately 57 kg of fuel consumed
- Sail-assisted configuration: approximately 39 kg of fuel consumed
- Fuel saving: approximately 32%

These results indicate the fuel saving potential attributable to sail-generated thrust under the simulated wind conditions, with both configurations maintaining constant engine rate throughout the voyage.

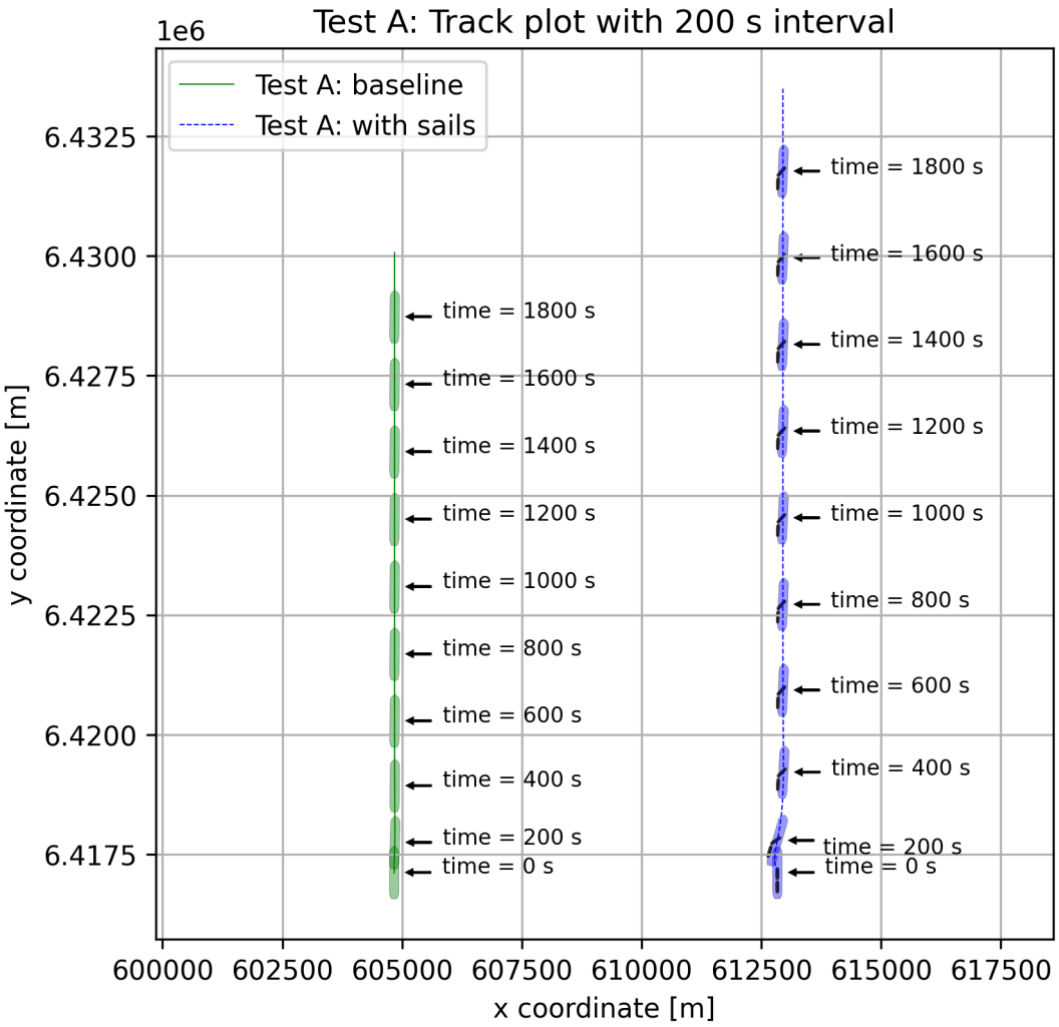


Figure 4 Ship tracks of Test A.

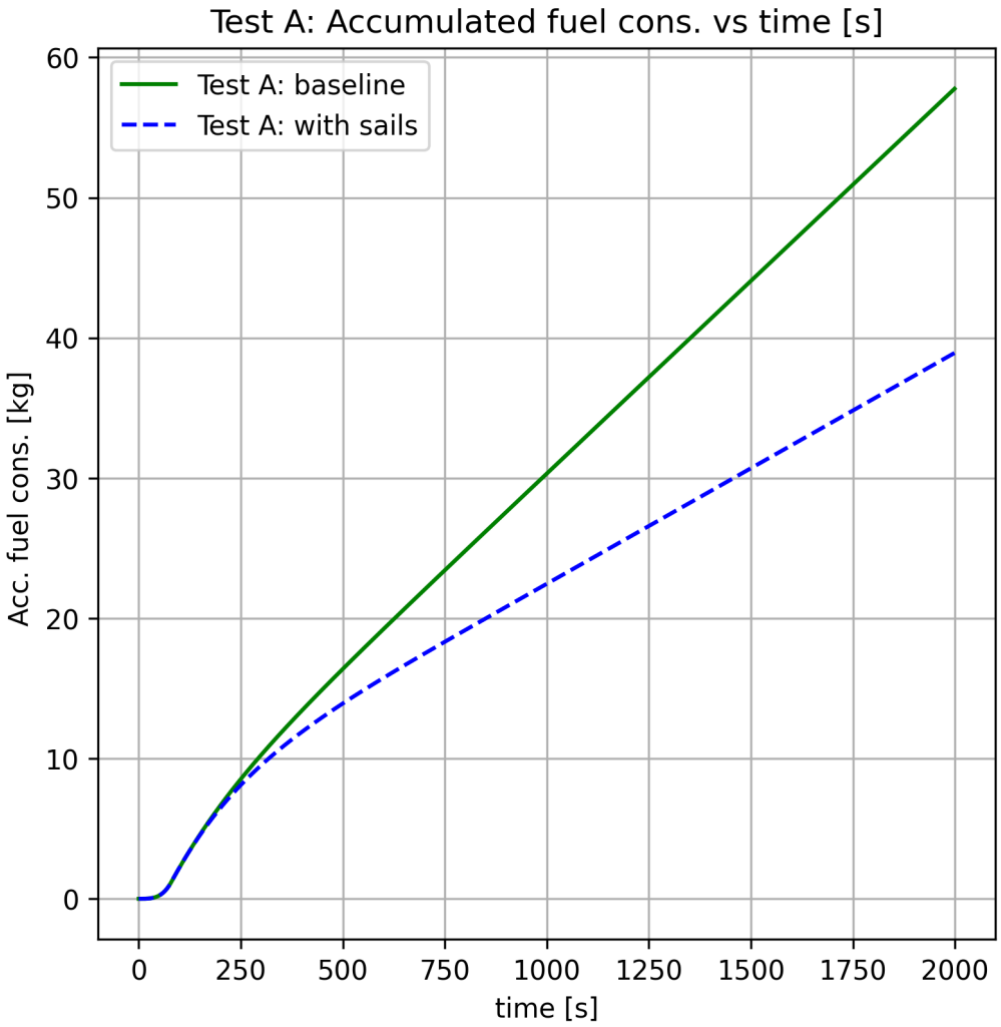


Figure 5 Accumulated fuel consumption of Test A.

Test B: Fuel Consumption Control Effectiveness

The results in Fig 6. And Fig 7. demonstrate that the fuel-saving capabilities of the propulsion control system effectively reduces fuel consumption while maintaining comparable voyage progress. Both configurations followed parallel northward courses over the 2000-second simulation, with similar distances covered.

The speed measurements (Fig. 8) show both configurations accelerating from approximately 2 m/s to steady-state, with the baseline reaching approximately 9 m/s while the fuel-saving mode stabilized at approximately 8.8 m/s.

The accumulated fuel consumption reveals the benefit of the fuel-saving mode:

- Baseline configuration (constant power): approximately 39 kg of fuel consumed

- Fuel-saving mode configuration: approximately 32 kg of fuel consumed
- Additional fuel saving: approximately 18% in exchange for a minor reduction in vessel speed

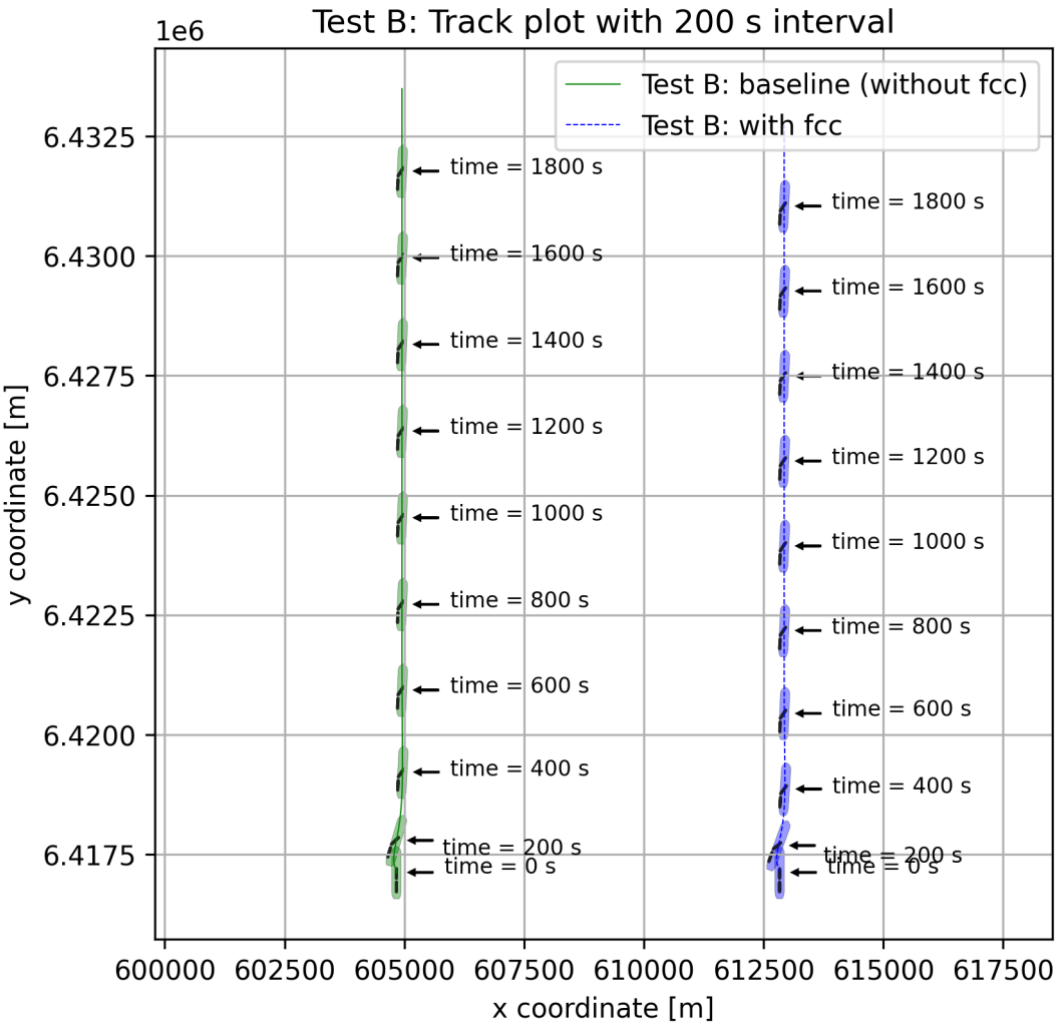


Figure 6 Ship tracks of Test B.

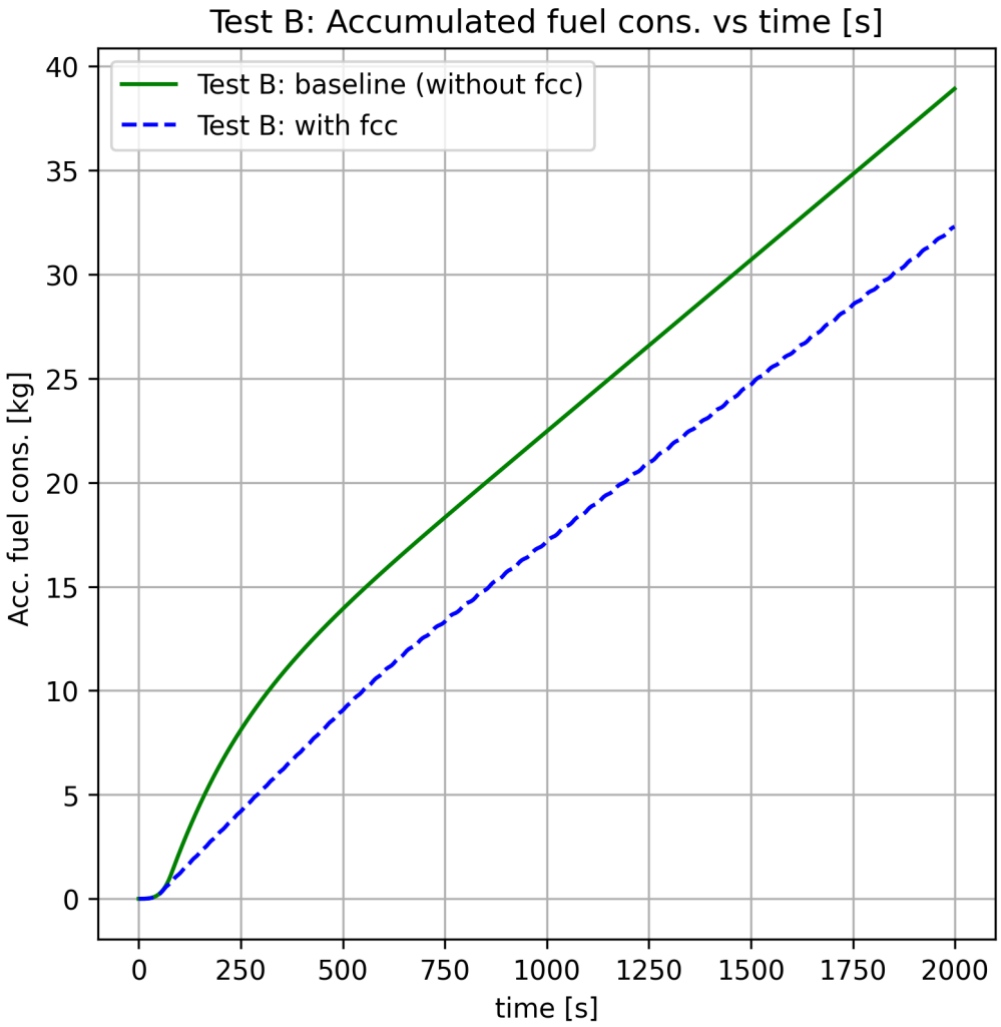


Figure 7 Accumulated fuel consumption of Test B.

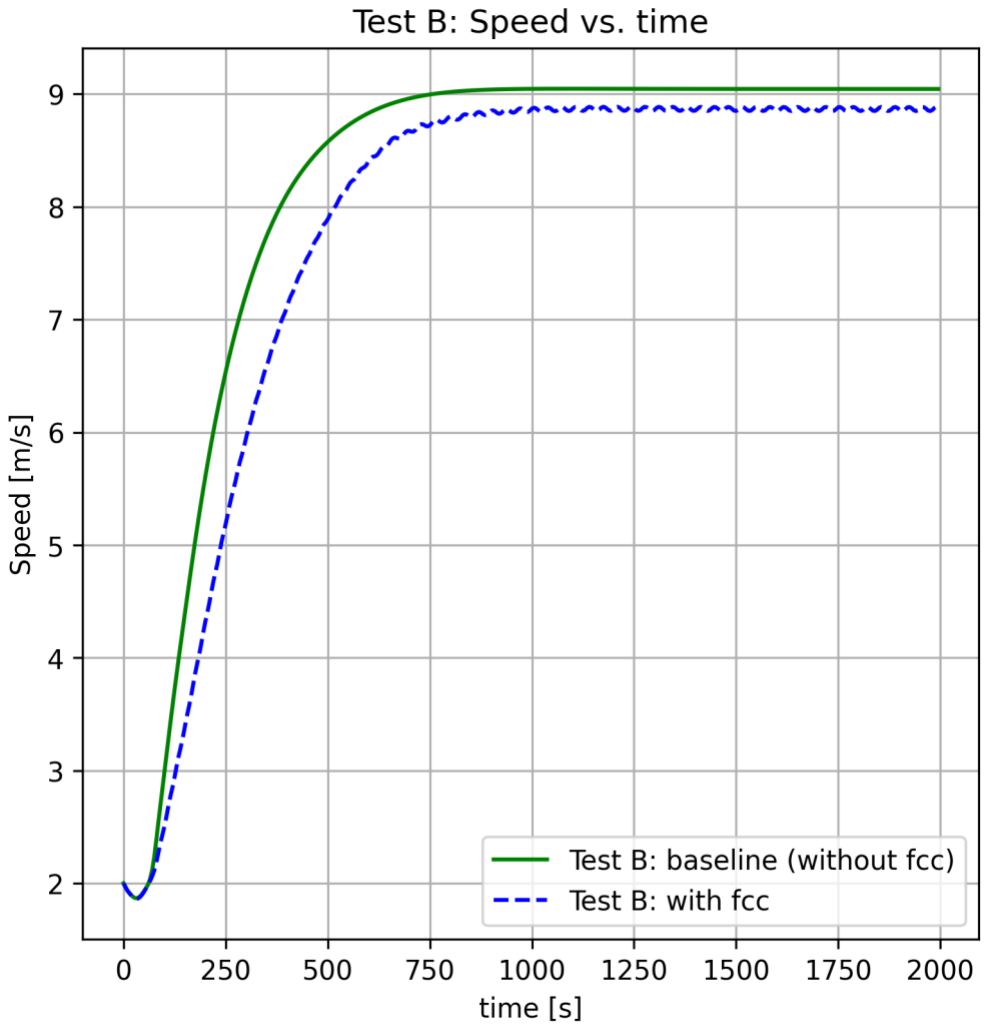


Figure 8 Ship speed of Test B.

Test C: Sail Control Response to Wind Changes

The results (Figs. 9 and 10) demonstrate that the sail control system correctly responds to changing wind conditions. The vessel maintained a steady northward course throughout the 2000-second simulation despite the wind direction change.

The boom angle response showed:

- Initial stabilization at approximately 30 degrees during the first 800 seconds
- Transient response to the wind direction change with a peak to approximately 70 degrees around 900 seconds
- Settlement to a new steady-state value of approximately -30 degrees by 1250 seconds

This behavior validates that the sail controller correctly repositions the sail to maintain the set angle of attack as the apparent wind angle changes.

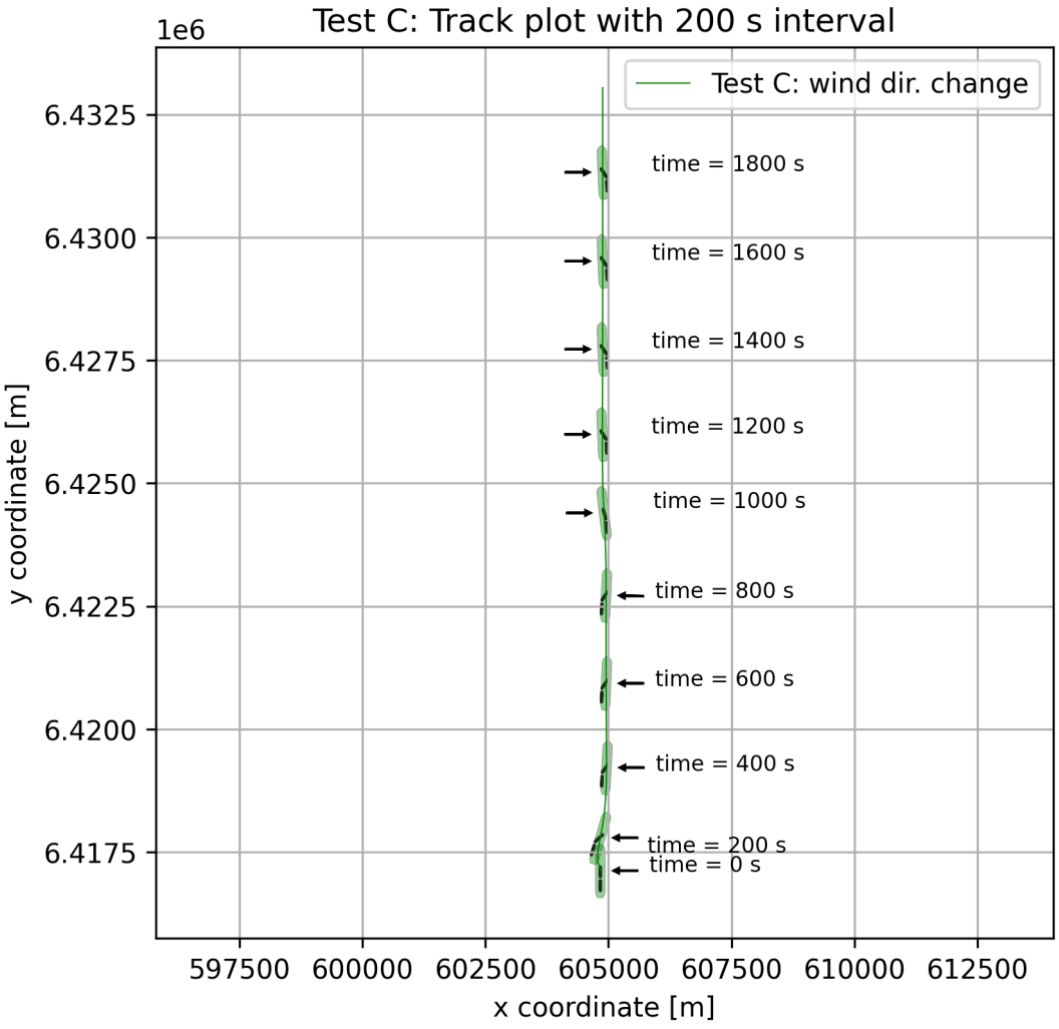


Figure 9 Ship track of Test C.

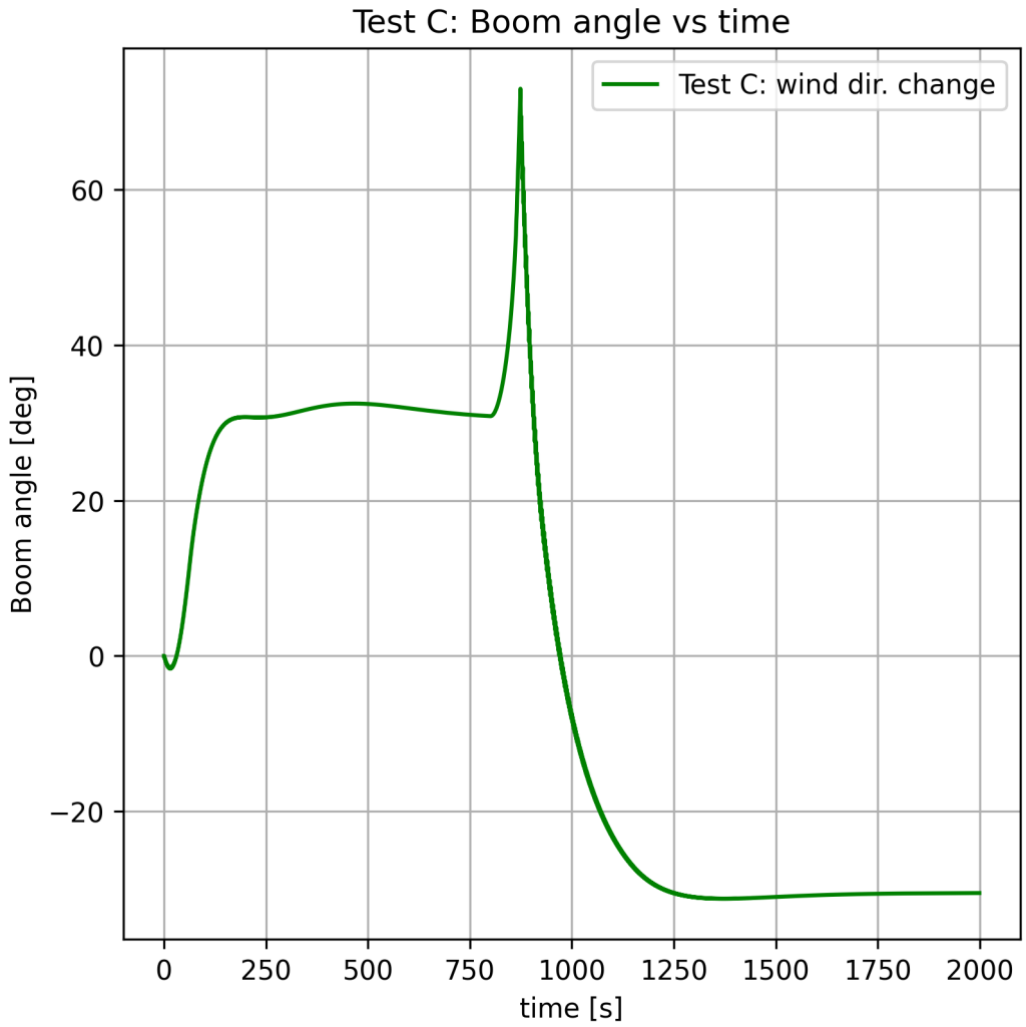


Figure 10 Boom angle of Test C.

Test D: Sail Control Response to Course Changes

The results (Figs. 11 and 12) indicate that the sail control system does not respond as expected to vessel course changes. The vessel initially sailed northward, then executed a 90-degree course change to port at approximately 800 seconds, continuing westward for the remainder of the simulation.

The boom angle response revealed unexpected behaviour after its initial stabilization at approximately 30 degrees during northward sailing (consistent with Test C) :

- Rapid increase to approximately 175 degrees during the course change maneuver between 800 and 1000 seconds
- Continuous short-amplitude oscillations following the maneuver

This behavior deviates from the intended operation, wherein the sail control system should adjust the boom angle to achieve the set angle of attack. The observations suggest that when sailing in running wind conditions, small variations in the apparent wind direction impede the effective operation of the sail control system.

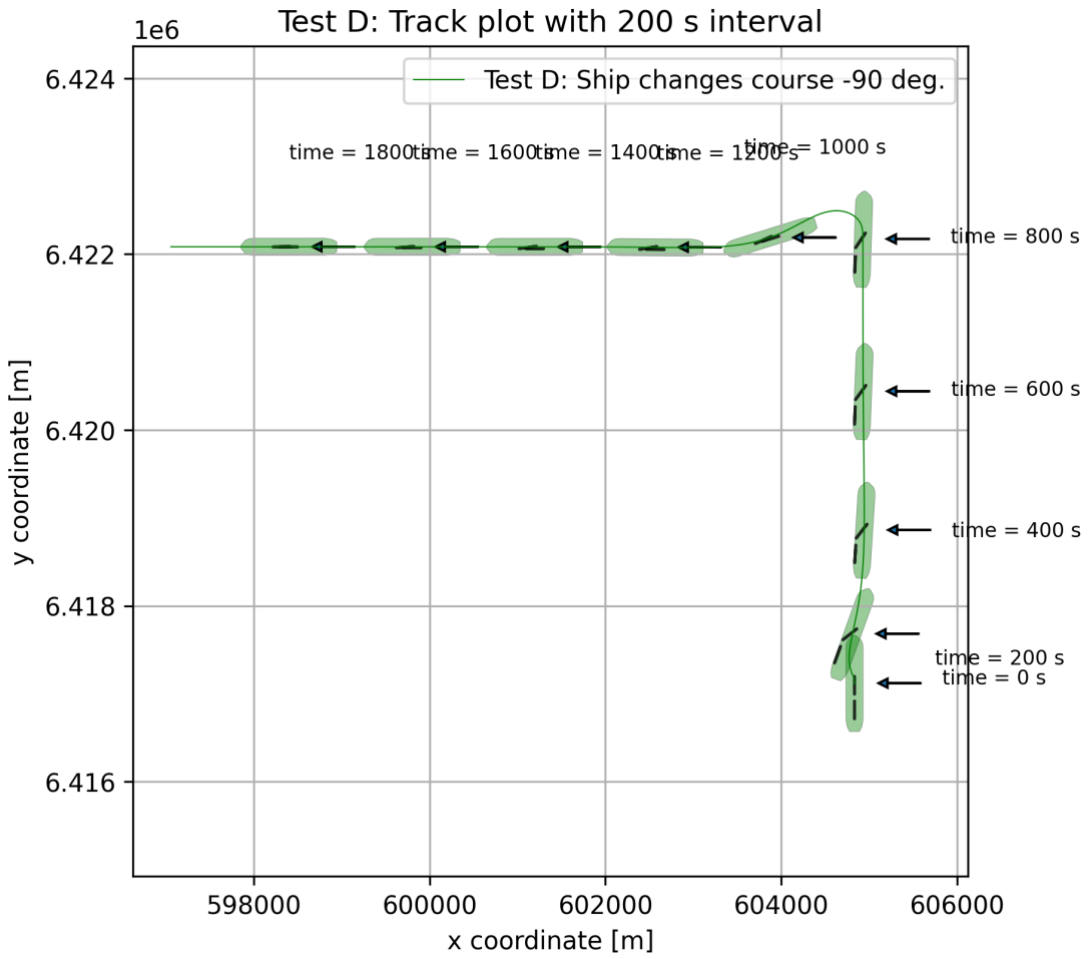


Figure 11 Ship track Test D.

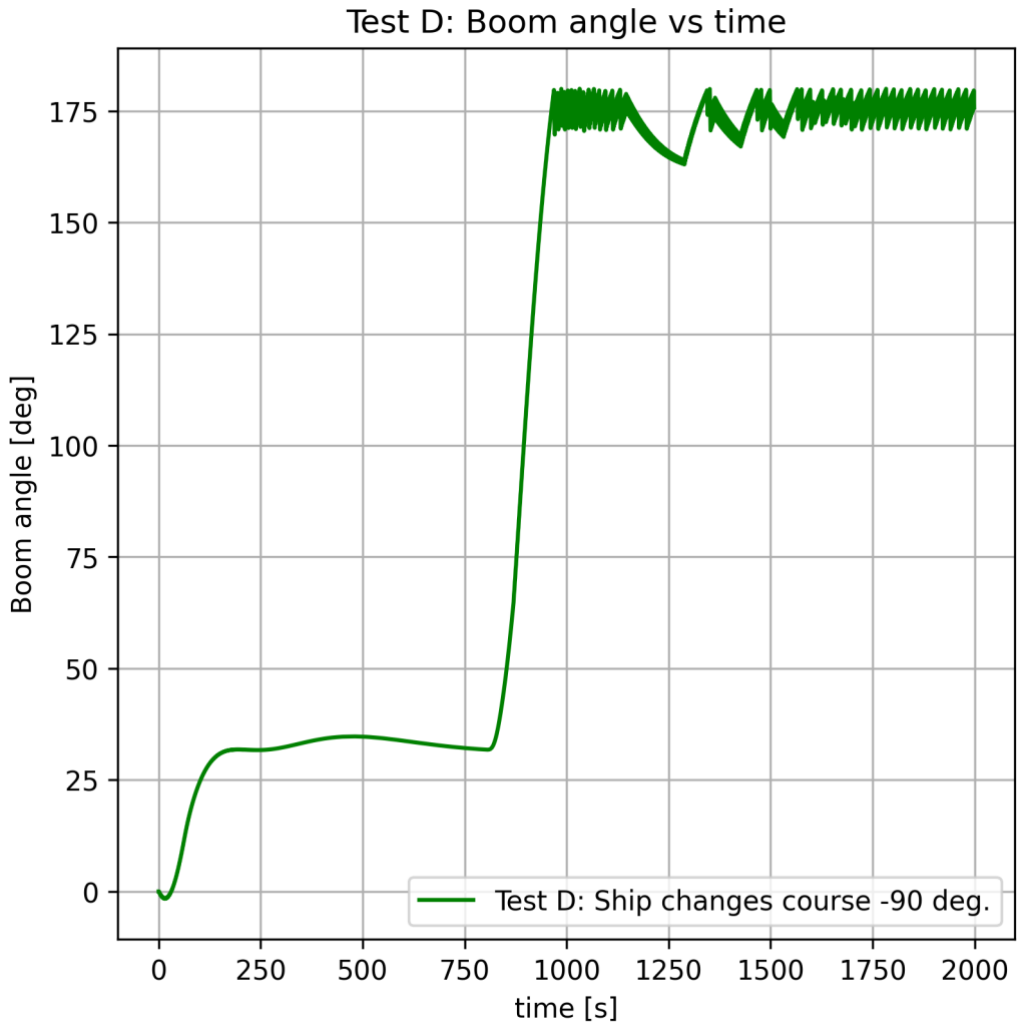


Figure 12 Boom angle Test D.

4.2 Discussion

The demonstration validated the core functionality of the distributed co-simulation platform. All four test cases executed successfully across the geographically distributed infrastructure, confirming that the Liaison tool [11, 12] enables real-time co-simulations over WAN connections.

Tests A and B demonstrated the expected fuel-saving benefits of wind-assisted propulsion and intelligent power management. Test C confirmed correct operation of the sail control system under changing environmental conditions.

Notably, Test D identified a deficiency in the sail control system's response to course changes. This finding illustrates a key benefit of the distributed co-simulation approach:

integration issues can be identified during the development phase, before physical installation, when corrections are more feasible. The platform enabled this early verification while respecting intellectual property boundaries, as each organization retained their models within their own infrastructure throughout the testing process.

5 Conclusions

This project successfully developed a distributed co-simulation platform that enables collaborative simulation across organizational boundaries while protecting intellectual property. The platform combines the FMI 3.0 standard for simulation model interfaces with the Zenoh communication protocol, addressing the gaps identified in existing solutions. The resulting open-source tool, Liaison [11, 12], allows organizations to participate in co-simulations without transferring their models to other parties, as all computations remain within each organization's own infrastructure.

The demonstration validated the platform's functionality through four test cases involving a wind-assisted cargo vessel. The simulations executed successfully across three geographically distributed sites connected via the internet, confirming that the architecture supports real-time co-simulations over wide-area networks. Tests A and B quantified fuel savings of 32% from wind-assisted propulsion and an additional 18% from the propulsion control system, while Test C verified correct sail control behavior under changing wind conditions.

The identification of a sail control system deficiency in Test D illustrates a key benefit of the distributed co-simulation approach: integration issues can be discovered during development, before physical installation, when corrections of the systems are significantly more feasible. The platform enables this early verification while respecting intellectual property boundaries, removing a significant barrier to collaborative system development in the maritime sector. By facilitating virtual testing of wind propulsion systems, the platform supports the transition toward sustainable shipping.

6 Future work

The Virtual Wind Ship project has established a foundation for distributed co-simulation, but several directions remain for continued development.

Extended FMI 3.0 support. The current Liaison implementation covers the subset of FMI 3.0 functions required for basic co-simulation. Future development should extend support to include the full FMI 3.0 Co-Simulation interface as well as the Scheduled Execution interface type, which would enable integration with virtual electronic control units and real-time execution environments. Support for FMI 3.0 Terminals would further simplify model connections by enabling semantic grouping of variables at the interface level.

SSP integration. The System Structure and Parametrization standard [13] provides a tool-independent format for describing complete simulation systems. Integrating SSP support into Liaison would enable standardized distribution of system configurations, that will simplify the setup of multi-organization co-simulations and improving reproducibility.

Hardware-in-the-loop integration. The platform has been developed to handle both SIL as well as HIL testing. However, within the project it has been used only for SIL testing. Future work should validate the platform's suitability for HIL applications, where strict timing requirements and deterministic communication become critical. This includes investigating latency characteristics across different network topologies and implementing mechanisms for guaranteed delivery and bounded response times.

Practical applications and use cases. The platform opens several near-term applications for the maritime industry:

- *Design phase collaboration.* Shipyards, equipment suppliers, and naval architects could jointly evaluate system integration during early design stages, reducing costly late-stage modifications. The sail control deficiency identified in Test D exemplifies how such issues can be caught before physical installation.
- *Virtual commissioning.* Before a vessel enters sea trials, the complete control system stack could be validated against high-fidelity virtual models of the ship and its environment, reducing commissioning time and risk.
- *Classification and approval processes.* Classification societies could participate in virtual testing campaigns without requiring access to proprietary models, potentially streamlining approval workflows for novel technologies such as wind-assisted propulsion.

- *Retrofit assessments.* When evaluating wind propulsion retrofits for existing vessels, shipowners could commission virtual integration studies where equipment suppliers demonstrate system compatibility without disclosing proprietary details.
- *Training and operational support.* The same distributed models used for design verification could feed into training simulators or decision support tools for voyage optimization.

Beyond wind propulsion. While this project focused on wind-assisted vessels, the platform addresses a general challenge in maritime system integration. Future applications could include hybrid and electric propulsion systems, autonomous vessel development, and integration of emerging technologies where multiple vendors must collaborate without exposing intellectual property.

7 References

[1] Modelica Association, "Functional Mock-up Interface Standard," [Online]. Available: https://fmi-standard.org

[2] Modelica Association, "System Structure and Parametrization (SSP) Standard," [Online]. Available: https://ssp-standard.org

[3] M. Krammer, M. Benedikt, T. Blochwitz, K. Alber, N. Amringer, C. Kater, S. Grunert, A. Heider, D. Beutlich, T. Neidhold, M. Chaker, C. Grassmann, and J. Klein, "The Distributed Co-Simulation Protocol for the Integration of Real-Time Systems and Simulation Environments," in *Proceedings of the 13th International Modelica Conference*, Regensburg, Germany, Mar. 2019, pp. 451-459.

[4] L. I. Hatledal, H. Zhang, A. Styve, and G. Hovland, "Co-simulation as a Fundamental Technology for Twin Ships," *Modeling, Identification and Control*, vol. 41, no. 4, pp. 297-311, 2020.

[5] DNV, Kongsberg Maritime, SINTEF, and NTNU, "Open Simulation Platform," [Online]. Available: https://opensimulationplatform.com

[6] L. I. Hatledal, "FMU-proxy," NTNU IHB, [Online]. Available: https://github.com/NTNU-IHB/FMU-proxy

[7] L. I. Hatledal, Y. Chu, A. Styve, and H. Zhang, "FMU-proxy: A Framework for Distributed Access to Functional Mock-up Units," in *Proceedings of the 13th International Modelica Conference*, Regensburg, Germany, Mar. 2019, no. 157, pp. 79-86.

[8] L. I. Hatledal, H. Zhang, K. G. Robbersmyr, and A. Styve, "A Language and Platform Independent Co-Simulation Framework Based on the Functional Mock-Up Interface," *IEEE Access*, vol. 7, pp. 109328-109339, 2019.

[9] L. I. Hatledal, "Ecos: An accessible and intuitive co-simulation framework," *Journal of Open Source Software*, vol. 10, no. 110, p. 8182, 2025, doi: 10.21105/joss.08182.

[10] Eclipse Foundation, "Zenoh: Zero Overhead Pub/Sub, Store/Query and Compute Protocol," [Online]. Available: https://zenoh.io

[11] Sanchez-Heres, L., Olsson, F., and Östh, J., "Liaison: an open-source tool for distributed co-simulations." In Modelica Conferences, pp. 639-644. 2025

[12] Liaison (2025). Liaison co-simulation tool. Code repository. URL: https://github.com/RISE/Maritime/liaison (visited on 2025-07-30).

[13] Modelica Association, "System Structure and Parameterization" Available: https://ssp-standard.org/