A Modelica-Based Cyber-Physical Simulation Architecture for Firmware-in-the-Loop Testing of Robotic Vehicles

American Modelica and FMI Conference 2026

Minhyun Cho, Minjoon Lee, Micah K. Condie, Michael P. Rooney,

Evan R. English, James M. Goppert, Arslan Khan,

Dongyan Xu, and Inseok Hwang

Purdue University and The Pennsylvania State University

PURDUE UNIVERSITY | School of Aeronautics and Astronautics

FD&C HSL PURDUE AERONAUTICS & ASTRONAUTICS

Outline

1 **Introduction**
Falsification objective and Modelica/FMU benefits — 3 - 5

2 **Simulation Architecture**
Board-time synchronization and model interfaces — 6 - 7

3 **FMI Export and Execution**
Rumoca compilation and the FMI 3.0 interface — 8 - 9

4 **Results**
Vehicle missions and runtime measurements — 10 - 13

5 **Discussion**
Limitations, fidelity-change testing and reachability — 14 - 16

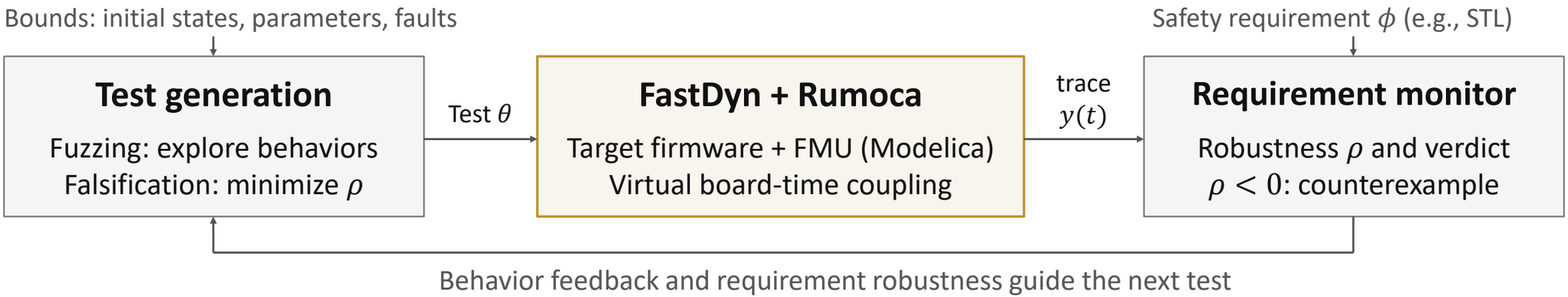
6 **Conclusion**
Contributions and remaining evidence — 17

Introduction

Specification-Guided Cyber-Physical Falsification and Fuzz Testing

Research goal: find system vulnerabilities and unsafe conditions/actions

Aims to provide firmware/plant execution for specification-guided tests of cyber-physical systems (CPSs)



Semi-formal verification Formal requirements + finite test search. Not a mathematical hard proof of the safety.

Volvo: development testing of electric car[1]

Continuous falsification of electric-machine and battery models to catch software defects.

Toyota: powertrain SIL and V&V[2]

Breach (falsifier) + C Bounded Model Checker (CBMC) exposed an injected, development-inspired fault in a production controller model.

[1] Eddeland, Johan Lidén, et al. "Enhancing temporal logic falsification with specification transformation and valued booleans." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 39.12 (2020)
[2] Yamaguchi, Tomoya, et al. "Combining requirement mining, software model checking and simulation-based verification for industrial automotive systems." 2016 Formal Methods in Computer-Aided Design (FMCAD). 2016.

Introduction

Why Modelica for Controlled Cyber-Physical Testing

Model-level changes make repeated experiments easier to control

Environment inputs and physical parameters remain explicit in reusable component models

Experiment change	Simulator-specific plugin workflow	Modelica-based workflow
Environment (e.g., wind, temperature)	Configure signals through backend APIs and model/plugin settings	Expose disturbance inputs and physical parameters in the model
System parameters (e.g., mass, inertia, delay)	Coordinate values across model descriptions and plugin code	Use named parameters and component hierarchies
Components fidelity (e.g., actuator / sensor)	Modify backend component logic and its integration as needed	Replace component equations and re-export with the same I/O contract
Fault activation (e.g., bias, fault signals)	Implement fault models and their responses in backend-specific code	Describe continuous response and discrete activation in equations

Change the physical model—not the testing infrastructure

Explicit equations, reusable components, and standardized FMI interfaces enable systematic variation of physical parameters, model fidelity, and fault dynamics without redesigning the firmware execution environment

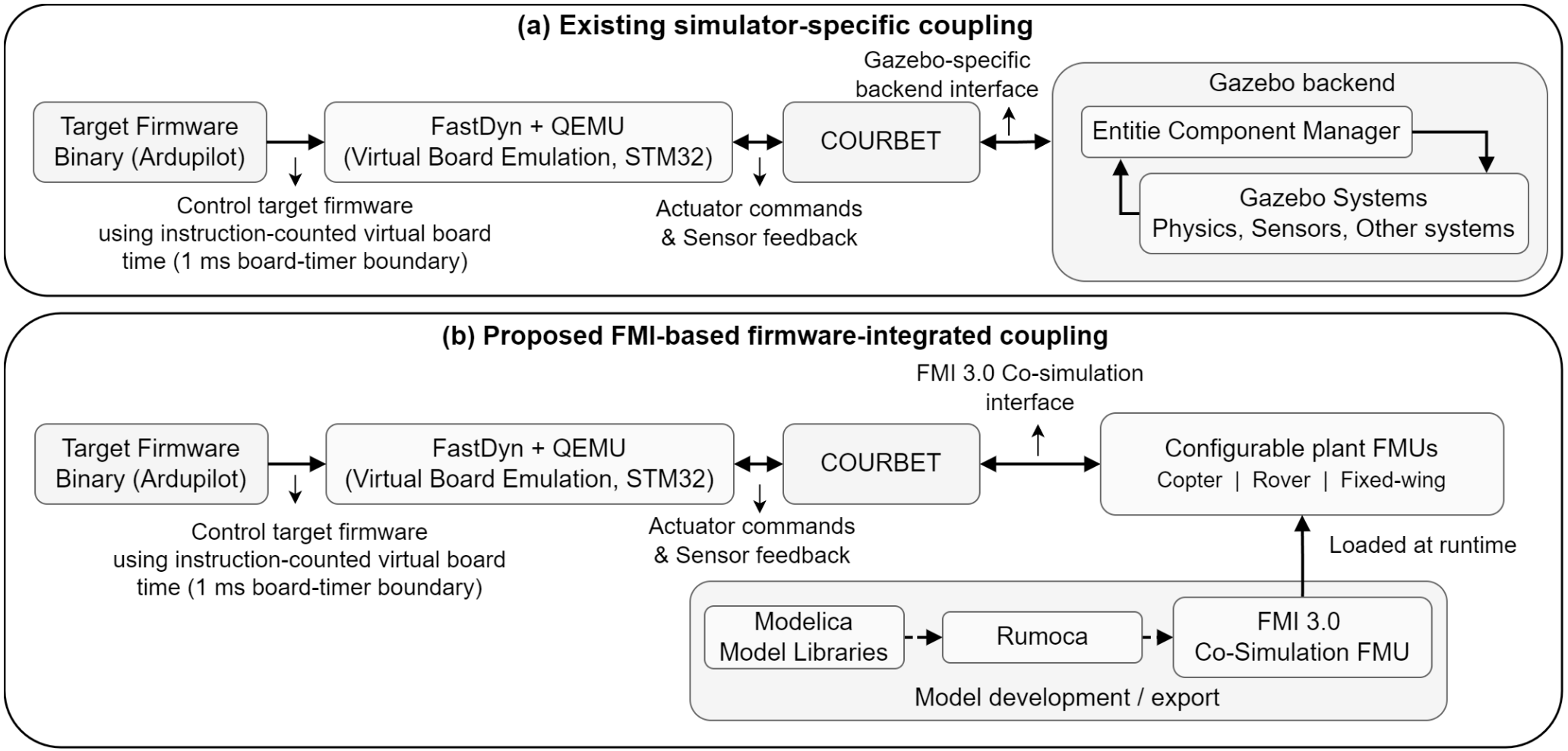
Reusable equations expose model assumptions and the variables under test

Introduction

From Simulator-Specific Coupling to FMI-Based Integration

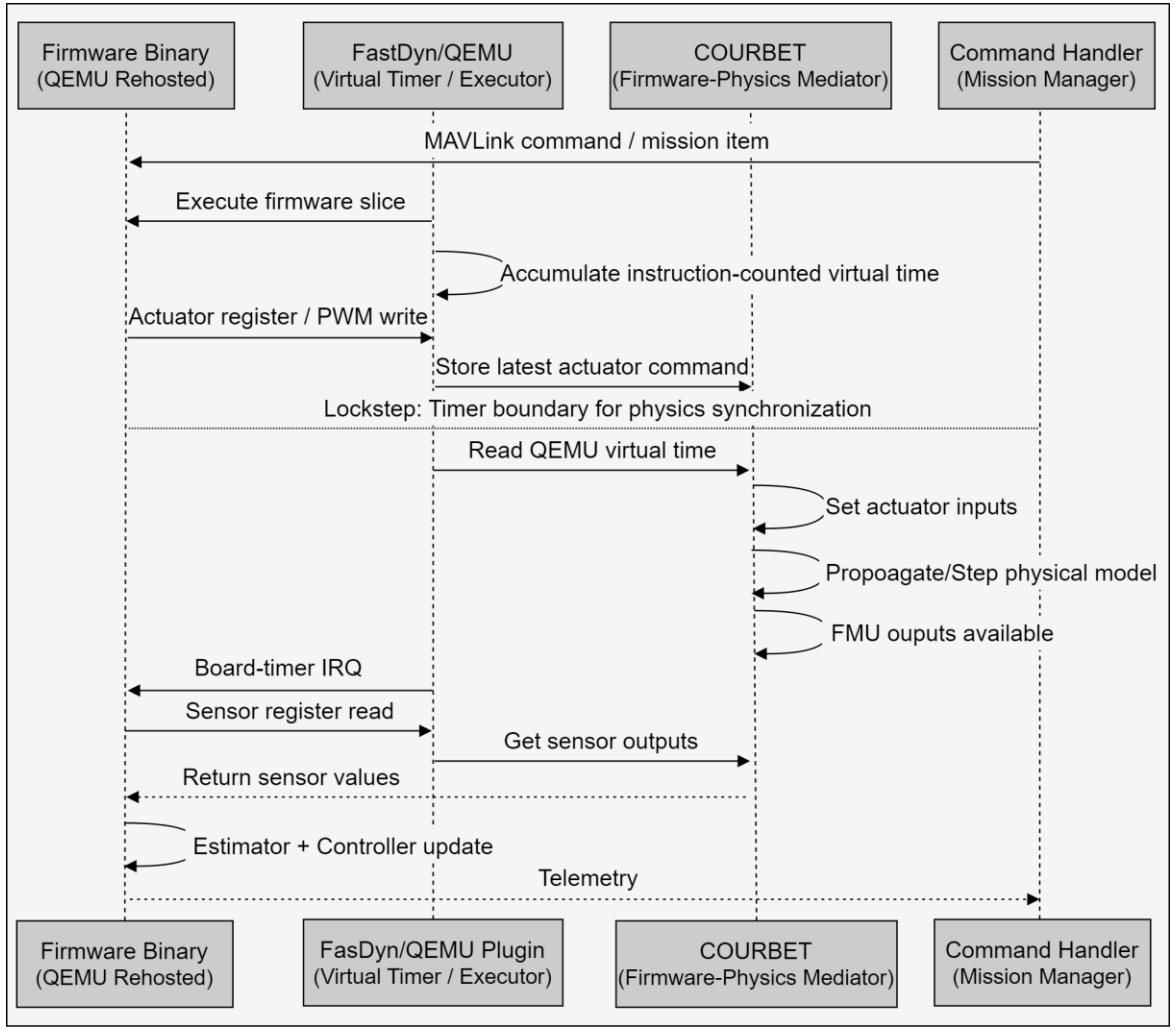
Modelica defines the plant & FMI standardizes how FastDyn executes it

Model changes stay behind one I/O contract. The target firmware and harness remain fixed



Simulation Architecture

Board-time Synchronization and Sampling Semantics



Board-time Synchronized Firmware-Plant Execution

Lockstep synchronization at every board-timer boundary:

$$h_k - t_{board,k} - t^{-}_{FMU,k}$$

$$t^{+}_{FMU,k} = t_{board,k}$$

$$e_k = t_{board,k} - t^{+}_{FMU,k} = 0$$

Nominal advancement: 1 ms, synchronization residual: 0.

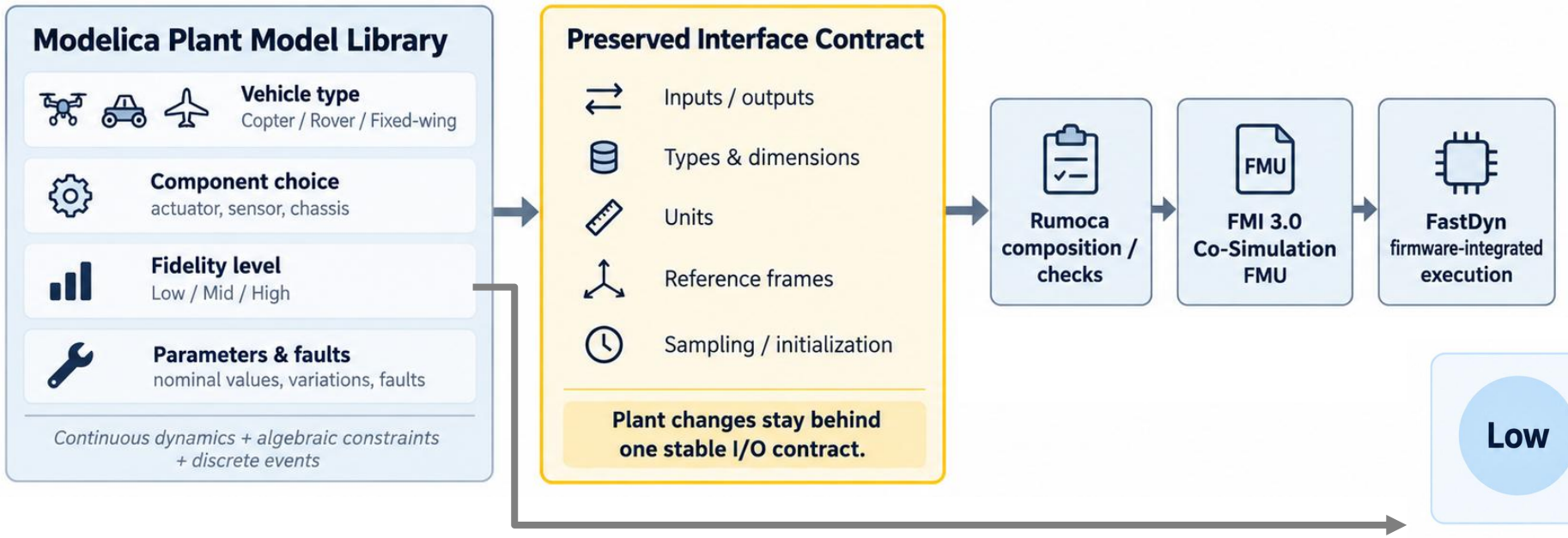
- Capture the latest actuator values.
- Advance the plant and publish sensors.
- Raise the timer IRQ, then resume firmware

Embedded-like Timing and I/O Semantics

Instruction-counted virtual time, periodic interrupts, and register-level I/O preserve the firmware execution sequence while synchronizing physical-model advancement

Simulation Architecture

Configurable Plant Models with Interchangeable Fidelity



Modelica enables component-level reconfiguration of physical dynamics while preserving system-level interfaces

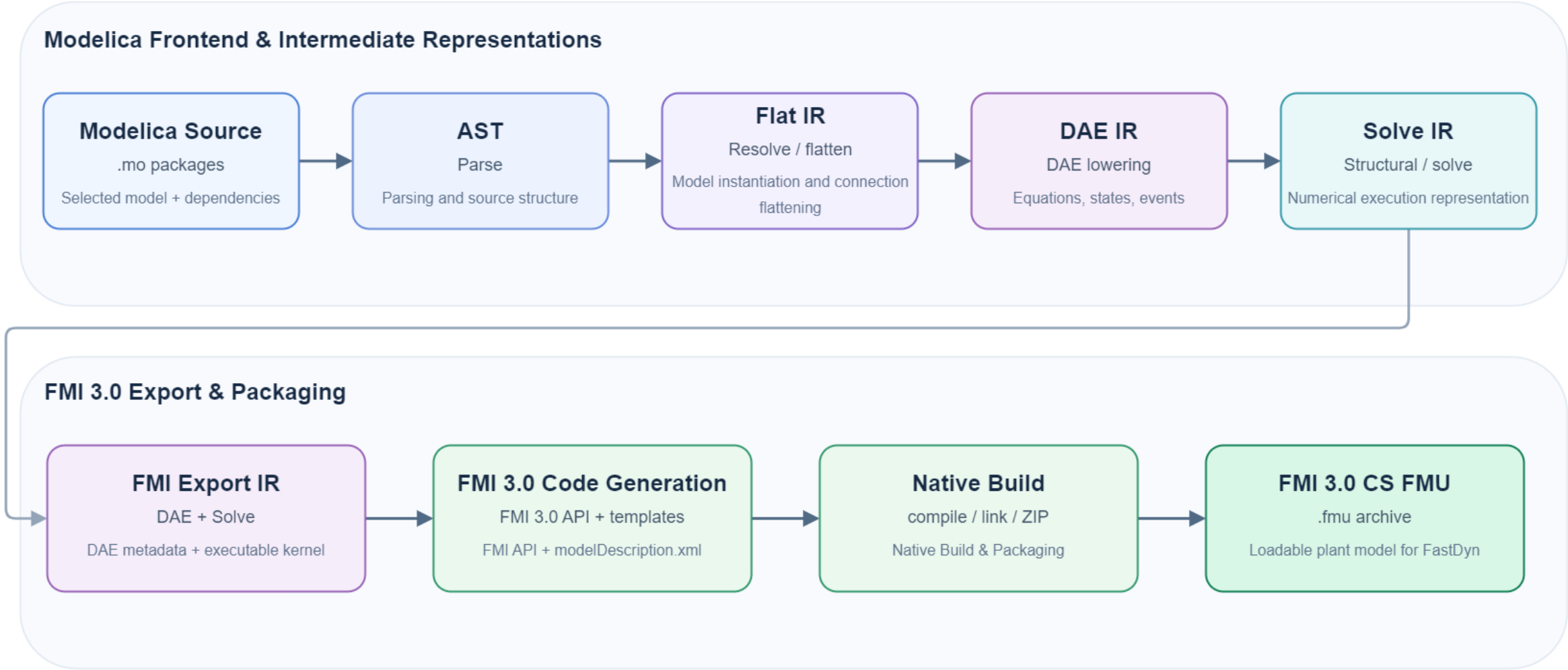
Hybrid physical model

Modelica represents continuous dynamics, algebraic constraints, and discrete events within a unified hybrid differential-algebraic equation (DAE) model

FMI Export and Execution

Rumoca: Modelica to FMI 3.0 compilation

Rumoca translates Modelica models into structured intermediate representations, generates FMI-compatible code, and packages executable FMUs for FastDyn



FMI Export and Execution

FMI 3.0 Features: Current Use and Planned Extensions

Current implementation: FMI 3.0 Co-Simulation with structured I/O.

Planned extensions: multi-rate scheduling, explicit event handling, and advanced simulation control

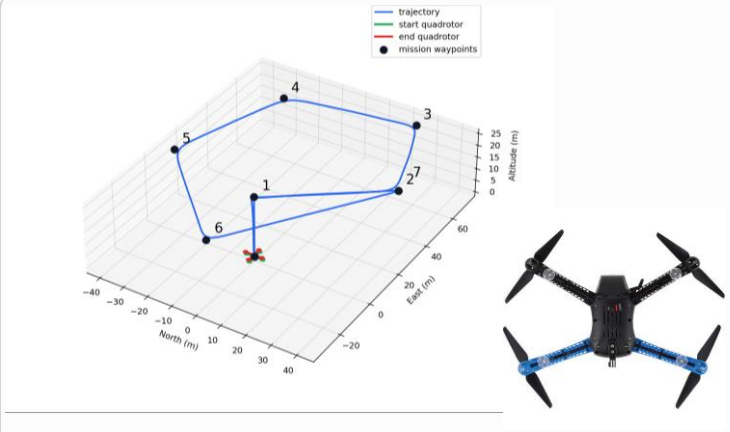
Feature	Description	Status in our framework
Co-Simulation API (also in2.0)	Exchange inputs/outputs and advance the plant with its solver	Used set / DoStep / get
Array variables (3.0)	Represent vector-valued I/O with explicit dimensions	Used PWM[4], quaternion[4]
Clocks and Scheduled Execution (3.0)	Synchronize periodic and triggered data updates and externally schedule model partitions	**Planned** **Multirate sensor modeling**
Event Mode (3.0 CS)	Explicit event handling across FMU boundary	**Planned** **Contact / switched dynamics**
Early return and Intermediate Update (3.0)	Enable early completion and intermediate communication	Not yet demonstrated
FMU state save / restore (optional)	Save and restore FMU state for repeated tests	Not yet demonstrated

Results

One Execution Framework, Three Vehicle Classes

Vehicle-specific firmware and plant FMUs share a common execution architecture.

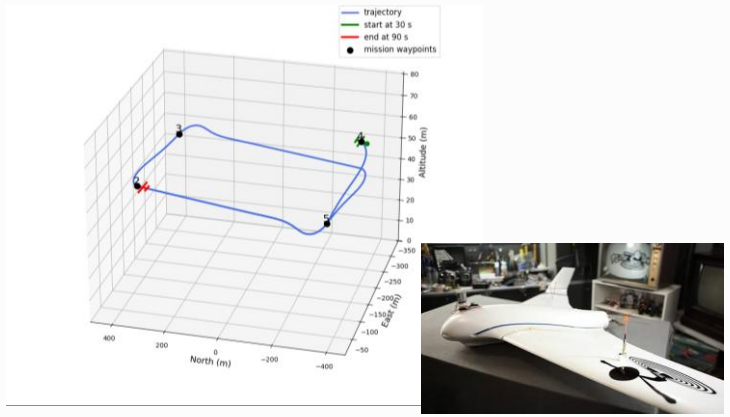
Firmware version: Ardupilot 4.6.2 (Copter / Plane / Rover)



ArduCopter

3DR Iris

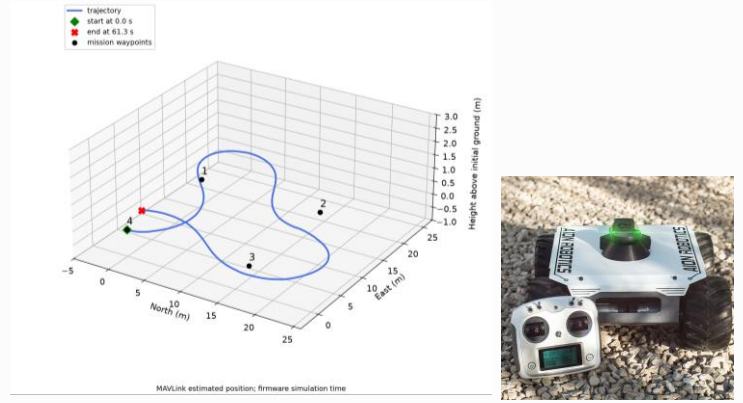
Takeoff → Waypoints → Landing



ArduPlane

Skywalker X8

Takeoff → Waypoints → Loiter → Landing



ArduRover

Aion R1

Waypoint Navigation

Shared Firmware-Plant Execution Pipeline

FastDyn / QEMU
ArduPilot firmware
⟷ COURBET ⟷ Plant FMU

Common STM32F427 virtual board environment · Same FMI-based coupling workflow

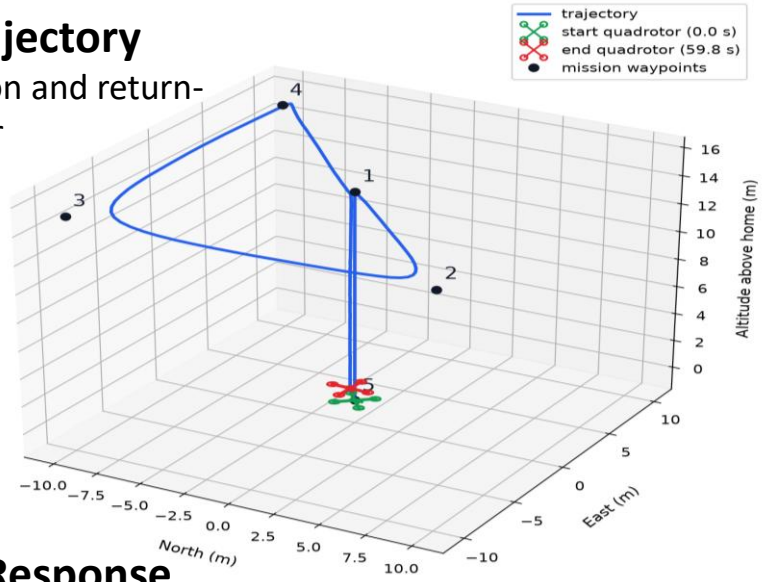
Results

ArduCopter Waypoint Mission in FMI-Based FITL

Closed-loop firmware–plant execution with waypoint tracking and landing.

3D Mission Trajectory

Waypoint navigation and return-to-ground behavior



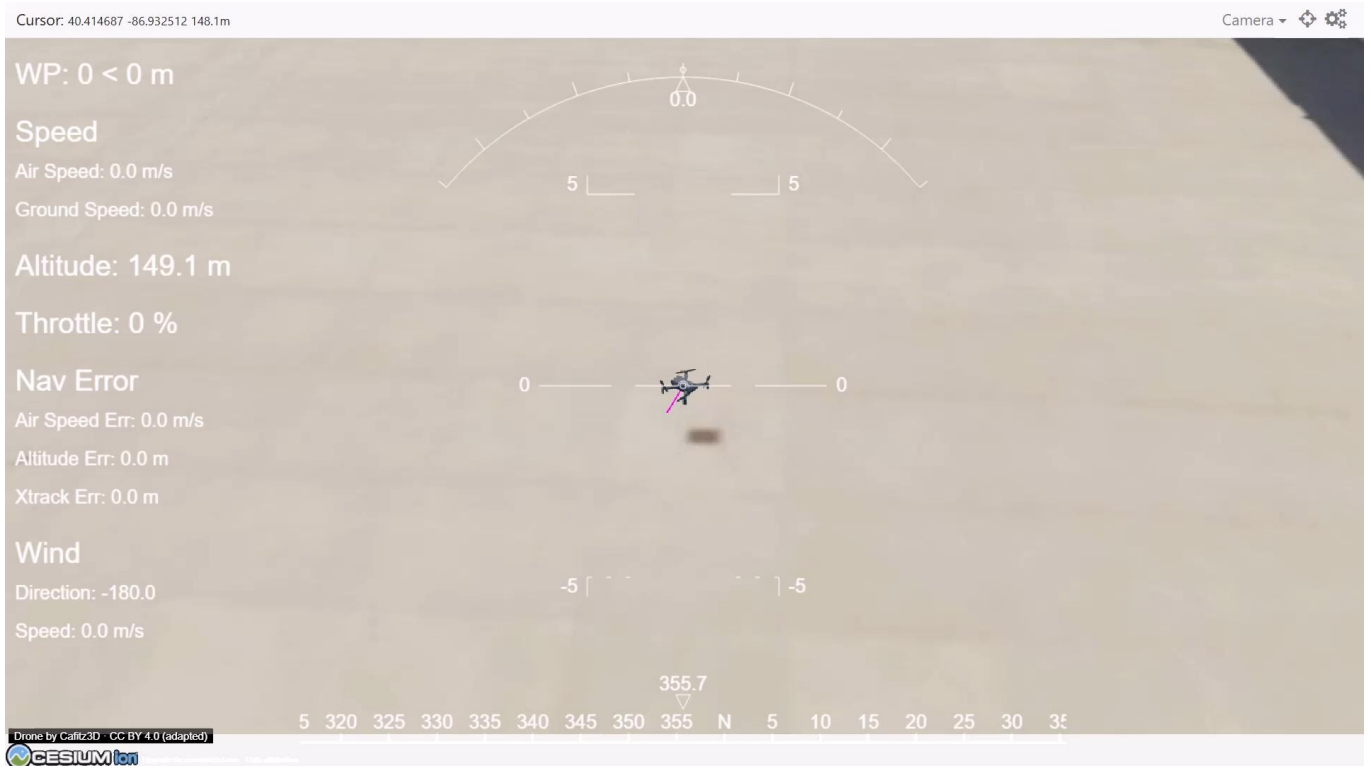
Plant Dynamic Response

Time histories of velocity, altitude, and attitude during the mission



MAVCesium Mission Replay

Recorded telemetry from the firmware-integrated simulation



Result

ArduPilot successfully executed a waypoint mission through the FMI-based firmware–plant interface.

Results

Sensor Attack Injection in FMI-Based FITL (ArduCopter)

Attack Scenarios

- **Baseline:** Nominal waypoint tracking without attack
- **GPS Spoofing:** Manipulated position measurements
- **Gyroscope Injection:** Injected bias and false angular-rate measurements

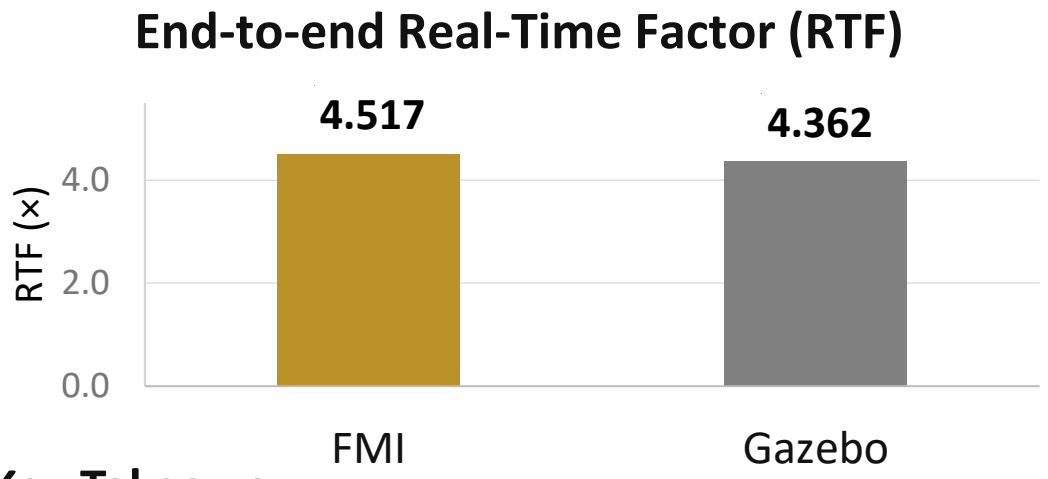
Key Takeaway

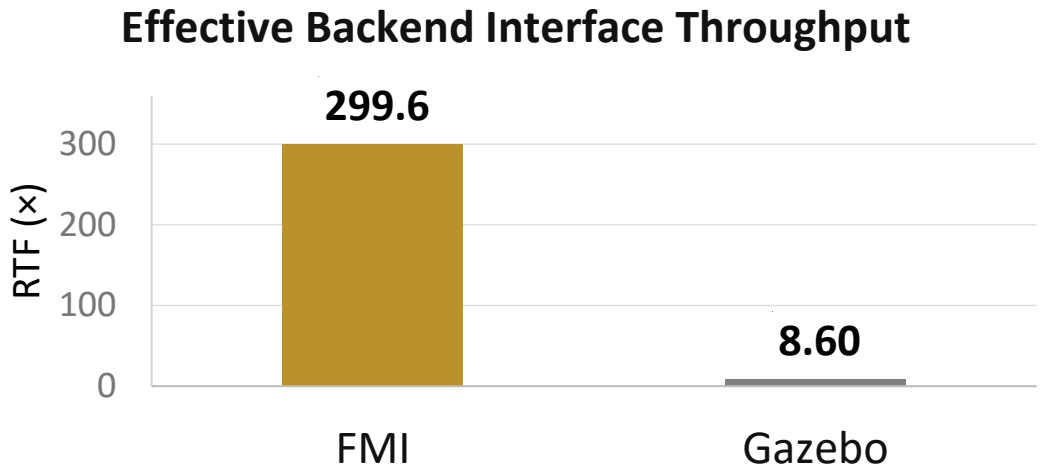
Modelica/FMI enables configurable, physics-informed attack models to be integrated with firmware-in-the-loop execution.



Results

Runtime Performance: End-to-End vs. Backend Throughput





Key Takeaway

The FMI-based backend provides a lightweight physical-model execution interface, although this throughput advantage translates into only a modest end-to-end speedup

Metric	FMI synchronous	Gazebo asynchronous
Backend advance interval	1 ms	9.333 ± 0.095 ms
Mean backend-call wall time	3.338 ± 0.138 µs	1085.339 ± 26.710 µs
Backend-call fraction	1.508 ± 0.065%	51.198 ± 0.604%

Test conditions:

- Five 60 s virtual-time runs. Table values show mean ± SD, Gazebo ran headless

Discussion

Current Limitations and Validation Priorities

Limitation	Implication for CPS Testing	Planned Improvement
Multi-fidelity integration in progress	Fidelity-dependent model substitution and testing remain limited.	Complete component integration and cross-fidelity testing.
Limited physical-model validation	Mission completion demonstrates functional execution, not quantitative physical accuracy.	Validate against reference data and quantify model discrepancies using conformal prediction.
Timing and peripheral fidelity	Board-time synchronization does not fully capture communication delays and asynchronous sensor updates.	Extend protocol-level timing and multi-rate sensor modeling.
Incomplete interface-contract validation	Physically incompatible signals may pass basic FMI type and unit checks	Extend interface contracts with automated frame and signal-semantic checks.
Falsification evaluation pending	Vulnerability discovery, fault coverage, and counterexample detection remain unevaluated.	Integrate STL-guided falsification and benchmark counterexample discovery.

The current framework demonstrates firmware-FMU execution across vehicle classes.
Future validation must establish whether configurable models preserve safety-critical behaviors and vulnerabilities.

Discussion

Future direction 1: Cross-fidelity fuzzing and falsification

Search cheaply, replay at higher fidelity, and preserve firmware and external I/O contracts.

Common physical scenario θ: initial conditions · plant parameters · sensor/actuator faults · activation timing

1 Discover · Low fidelity

- Explore bounded states, parameters, and faults.
- Guide search with STL robustness and coverage.
- Retain candidate failures θ^*.

Output: candidate scenarios θ^*

2 Replay · Mid / High

- Keep firmware and the FMI I/O contract fixed.
- Map θ^* to model-compatible states and inputs.
- Replay θ^* with mid/high FMUs.

Output: replay traces

3 Compare · Preservation

- Compare robustness and violation signs.
- Inspect trajectory and fault responses.
- Flag model-sensitive failures.

Output: cross-fidelity agreement

Example STL safety property

$$\varphi = G_{[0,T]}(|e_\phi(t)| \leq e_{\phi,max}) \qquad \rho_\phi(M,\theta) = \min_{t\in[0,T]}(e_{\phi,max} - |e_\phi(t;M,\theta)|)$$

e_ϕ = roll angle tracking error, $e_{\phi,max}$ = maximum allowable roll angle tracking error, $\rho_\phi < 0$ indicates that the violation of the specified roll-angle tracking performance requirement for controllers, M indicates model configuration.

Discussion

Future direction 2: Complementary reachability analysis

Screen test regions with reachability, then prioritize costly firmware–FMU trials.

Common physical scenario θ: initial conditions · plant parameters · sensor/actuator faults · activation timing

1 Abstract · closed loop

- Modelica plant / hybrid IR
- Bounded controller and timing abstraction
- Initial sets, inputs, parameters and faults

Output: analysis-ready model

→

2 Screen · reachability

Conservative reachable enclosure $R(\theta)$

SAFE · disjoint → prune θ

OVERLAP · unknown → prioritize

Output: ranked test regions

→

3 Test · concrete FITL

- ArduPilot binary + plant FMU
- Replay prioritized test scenarios
- Monitor STL robustness and violations

Output: model counterexamples

Conditional safe pruning $R(\theta) \subseteq \hat{R}(\theta), \hat{R}(\theta) \cap (\text{Unsafe}) = \emptyset \implies \text{prune } \theta$

$R(\theta)$: states reachable over $[0, T]$, $\hat{R}(\theta)$: sound outer approximation of true reachable set $R(\theta)$

- No intersection: certification of safety under the modeling assumptions and given conditions
- Intersection: inconclusive and motivates targeted concrete FITL trials to search for realizable counterexamples

Conclusion

Reusable and Configurable FMI-Based Firmware-in-the-Loop Simulation

1. Decoupled Firmware - Plant Integration

- Replaced simulator-specific plant coupling with a standardized Modelica/FMI interface, facilitating systematic cyber-physical testing and verification.

2. Multi-Vehicle Support and Configurable Physical Models

- Demonstrated a common FITL workflow for ArduCopter, ArduRover, and ArduPlane (ArduPilot 4.6)
- Extending support for component substitution and multi-fidelity model configuration

3. Efficient Firmware-Integrated Simulation

- Achieved a modest end-to-end RTF improvement (4.517× FMI vs. 4.362× Gazebo).
- Measured **34.8× higher effective backend interface throughput** with FMI-based execution.

Future Work and Validation

- **CPS Falsification:** Cross-fidelity fuzzing and reachability-guided counterexample discovery.
- **Model Validation:** Quantitative tracking, fault-response consistency, and repeatability across fidelity levels.
- **Integration Reliability:** Automated frame consistency and result traceability checks.

Acknowledgment

Thank you.

Questions & Discussion

Repositories
Rumoca: https://github.com/CogniPilot/rumoca
FastDyn: https://github.com/PSecLab/FastDyn
modelica-models:
https://github.com/CogniPilot/modelica_models
fire-modelica-models:
https://github.com/mhcho1994/fire_modelica_models

Research program

DARPA FIRE

(Faithful Integrated Reverse-engineering and Exploitation)

This work is based on research sponsored by Defense Advanced Research Projects Agency (DARPA) under agreement number FA8750-24-2-0500. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.