RVSPEC: Cyber-Physical Interplay Graphs for Formal Specification of Robotic Vehicle Control Software

Chaoqi Zhang[1], Minhyun Cho[2], Inseok Hwang[2], and Hyungsub Kim[1]

Abstract— **Robotic vehicles (RVs) have increasingly deployed in critical missions. Yet, RV control software is prone to logic bugs that cause unexpected physical behaviors, deviating from the developers' intentions. For instance, Hakuto-R Mission 1 lunar lander physically crashed on the lunar surface due to a misinterpretation of sensor data. To discover such bugs, developers leverage bug-finding tools, from formal methods to fuzzing. To use these tools, human experts first need to manually create formal specifications (e.g., temporal logic) as bug oracles. Yet, such manual efforts are time-consuming and error-prone. Previous efforts to automatically generate such specifications merely translate natural-language documentation into formal specifications. In turn, they overlook the *cyber-physical interplay* inherent in RVs, which is often absent from the documentation, e.g., altitude changes caused by air pressure and servo lag.**

To tackle this limitation, we introduce RVSPEC, an automatic specification generation framework. It first constructs a *cyber-physical interplay graph* (CPG). It captures the quantification about how much internal (control software-dependent and hardware-specific properties intrinsic to an RV) and external factors (environmental conditions) influence the RV's physical states. Then, RVSPEC uses the CPG to guide large language model agents, enabling the generation of *cyber-physical interplay–aware formal specifications*. We evaluated RVSPEC on four popular RV control software packages, including ArduPilot and PX4 for aerial vehicles, openpilot for autonomous vehicles, and cFS for spacecrafts. The evaluation showed that specifications created by RVSPEC achieved an accuracy of 80.7%, whereas the baseline's ones attained 51.6%. When applying the specifications for fuzzing, those generated by RVSPEC reduced the number of false positives from 4,790 (baseline) to 964 (79.9% reduction) while preserving the bug-finding capability. The code is available at https://github.com/KimHyungSub/RVSpec.

I. INTRODUCTION

Robotic vehicles (RVs)—including drones, autonomous vehicles, and spacecrafts—have increasingly been deployed in critical missions (e.g., search-and-rescue, military operations). Yet, RV control software is prone to software bugs, particularly logic bugs. These bugs cause unexpected physical behaviors that deviate from the developers' intentions without involving memory access violations (e.g., buffer overflows). For instance, Hakuto-R Mission 1 lunar lander physically crashed on the lunar surface due to a misinterpretation of sensor data [1]. The fundamental root causes of such logic bugs are due to the difficulty of tracing how changes in code translate into behavioral modifications in the physical environment [2]. In fact, a prior work [3] discovers that 98.2% of bugs in two popular control software packages (ArduPilot [4] and PX4 [5]) are logic bugs, and only 1.8% of bugs are non-logic bugs (i.e., memory bugs).

To detect logic bugs, developers have leveraged various methodologies, from formal methods [6] to fuzzing [7], [8]. All of them rely on formally defined specifications of physical behaviors, expected by developers, as bug oracles.

Yet, developers normally document specifications using natural language rather than formal languages (e.g., temporal logic [9]). Indeed, our survey of 20 widely used open-source RV control software projects discovers that 19 out of the 20 projects use natural language in their specifications[1]. Only F Prime [10] for spacecraft control software uses the formal language (F Prime Prime) [11]. Thus, to use the bug-finding tools, human experts first need to manually create formal specifications based on documentation and their domain knowledge, which is time-consuming and error-prone.

For instance, the authors of PGFuzz [8] manually created 56 metric temporal logic (MTL) formulas as formal specifications. Yet, the two authors spent a full day of manual effort. Given that developers keep updating the software, manually creating formal specifications is not sustainable. Moreover, as we will detail in Section VII-B, our analysis and experiment discover that 29 out of the 56 MTL formulas (51.8%) are inaccurate to capture actual physical behaviors of RVs[2].

To automate the formal specification generation, recent works have explored using large language models (LLMs) [12]. Yet, these methods are primarily designed for general-purpose software (e.g., Java utility programs). Thus, their specifications overlook the *cyber-physical interplay* inherent in RVs—such as altitude changes caused by air pressure, servo lag, and deceleration ramping.

In particular, *internal and external factors* play key roles in the *cyber-physical interplay*. In here, we define *internal factors* as control software-dependent and hardware-specific properties intrinsic to the vehicle (e.g., sensor fusion, mass, response delay), and *external factors* as environmental conditions outside the vehicle's control (e.g., wind, temperature, magnetic disturbance) that influence its physical behavior.

For instance, a manually extracted specification in PGFuzz [8] describes expected physical behaviors during landing flight mode: "*The vehicle should descend at* 1.0 *m/s when* 10 *meters above altitude, and at* 0.5 *m/s when* 10 *meters below altitude*". At a high level, this description is seemingly correct. Yet, an instantaneous change in vertical speed exactly

Corresponding author: Hyungsub Kim

[1]Chaoqi Zhang (cz42@iu.edu) and Hyungsub Kim (hk145@iu.edu) are with the Department of Computer Science at Indiana University Bloomington

[2]Minhyun Cho (cho515@purdue.edu) and Inseok Hwang (ihwang@purdue.edu) are with the School of Aeronautics and Astronautics at Purdue University

[1]https://tinyurl.com/4u8mzyfa

[2]https://tinyurl.com/5dbpcm84

at the altitude of 10 meters is physically infeasible due to the aforementioned *cyber-physical interplay*. In practice, a drone decelerates gradually over a finite period due to *internal factors* such as servo lag, control loop delay, and deceleration ramping. Further, *external factors* like wind and temperature can further affect the drone's vertical speed. Without considering these factors, the specifications misrepresent an RV's physical behaviors.

Two fundamental challenges hinder the automatic generation of formal specifications for RVs. First, there is no systematic way to quantify how *internal and external factors* influence physical states of RVs (e.g., position, attitude). Without this understanding, it is difficult to model physical transitions appropriately. Second, documentations in natural language often omits detailed contextual knowledge about a vehicle's physical behavior—for instance, a drone decelerates gradually over a finite period rather than experiencing instantaneous speed reduction. Thus, prior works [12] are limited by their exclusive reliance on textual documentation.

To address these challenges, we introduce RVSPEC, an automatic specification generation framework. It constructs a *cyber-physical interplay graph (CPG)*, which captures how much *internal and external factors* influence the physical states of RVs. RVSPEC builds this graph using simulations and/or real-world tests. Then, RVSPEC uses the CPG to guide LLM reasoning with context that is not explicitly stated in the documentation, enabling the generation of *cyber-physical interplay–aware formal specifications*.

We evaluate RVSPEC on four popular RV control software packages, including ArduPilot and PX4 for aerial vehicles, openpilot [13] for autonomous vehicles, and cFS [14] for spacecrafts. The specifications created by RVSPEC achieve 80.7% accuracy compared with 51.6% baseline. When we use the specifications to run fuzzing, those generated by RVSPEC reduces the number of false positives by 79.9% from 4,790 to 964 while preserving the bug-finding capability.

Overall, the CPG provides insights into how *internal and external factors* influence an RV's physical states. In turn, the formal specifications automatically created from the CPG enable developers to avoid expending time and effort on investigating false-positive buggy behaviors in their software.

In summary, we make the following contributions:

- To the best of our knowledge, we are the first to introduce the *cyber-physical interplay graph (CPG)* as a representation that captures how *internal and external factors* interact with physical states of RVs.
- We design RVSPEC, a multi-agent LLM-based framework that leverages both documentation and CPG to generate *cyber-physical interplay–aware specifications*.
- We evaluate RVSPEC on four RV control software for aerial vehicles, autonomous vehicles, and spacecrafts. The evaluation shows that RVSPEC creates specifications with 80.7% accuracy outperforming the 51.6% (baseline). When we use the specifications for fuzzing, RVSPEC reduces false positive cases by 79.9% from 4,790 to 964 while maintaining the bug-finding capability.

II. Related Work

Traditional Natural Language Processing (NLP)-based Methods. [15], [16] rely on NLP techniques (e.g., heuristic rule, transformer-based model) to merely translate natural language into formal specifications. Thus, they do not consider *cyber-physical interplay* (i.e., *internal and external factors*). It leads to specifications that are syntactically correct but fail to capture physical behaviors of RVs in the real world. For instance, dynamically changed air pressure can increase barometer noise and cause a drone's altitude to fluctuate. Yet, since this is not explicitly documented, NLP-based methods fail to capture this disturbance.

LLM-based Methods. LLM-based approaches leverage direct translation methods [12], domain-specific adaptations [17] for traffic rules, robotic task, and advanced architectures including multi-stage synthesis [18]. Yet, these LLM-based approaches also overlook the *cyber-physical interplay* inherent in RVs, focusing merely on syntactic translation.

III. Background

Formal Methods and Fuzzing. Developers use formal methods and fuzzing techniques to discover logic bugs in software. Formal methods require two different inputs: (1) models to specify software in the form of finite state machines and (2) formal specification (e.g., temporal logic). They provide mathematical guarantees whether the models violate the specification via tools like model checking [6]. On the contrary, fuzzing tests specific software inputs at runtime [7], [8] to check whether the executed inputs violate the specifications. Given that both of formal method and fuzzing require formal specifications, creating accurate specifications plays a key role in finding bugs in software.

Metric Temporal Logic (MTL). MTL is a formal specification language for specifying system behaviors over time. This paper focuses on MTL rather than signal temporal logic (STL), following prior works [7], [8] that leverage MTL to identify logic bugs in control software, which allows us to directly compare RVSPEC with the previous works.

MTL consists of propositional logic—which includes conjunction ($\wedge$), disjunction ($\vee$), and negation ($\neg$)—with temporal operators such as $\Box$ (always), $\Diamond$ (eventually), and U (until). MTL defines formulas as $\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \; U_I \; \varphi_2 \mid \Box_I \varphi$, where I is a real-time interval. For example, a MTL formula: $\Box \; \{(\text{Mode}_t = \text{FLIP}) \rightarrow (\Diamond_{[0,2.5]} \text{Mode}_t = \text{FLIP})\}$ specifies "whenever the drone enters `FLIP` flight mode, it must transition to `FLIP` mode within 2.5 seconds".

IV. Motivating Example

Landing is a common phase in drone operations. According to ArduPilot's documentation [19], it operates as follows:

- Descends to 10 m using the regular altitude hold controller, which descends at the speed specified by the `WPNAV_SPEED_DN` configuration parameter.
- Below 10 m, the drone must descend at the rate specified by the `LAND_SPEED` parameter.

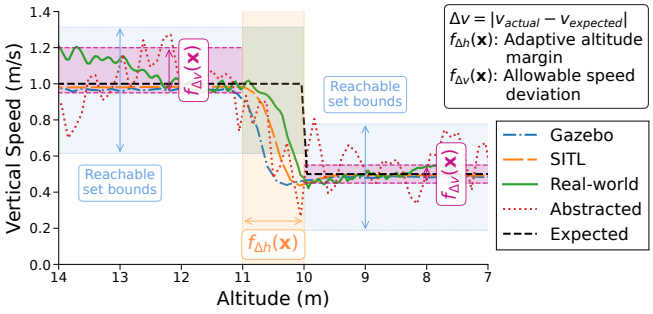


Fig. 1: Vertical speeds during landing, compared across real-world, SITL, Gazebo, and abstracted controller model simulations. The CPG-enhanced annotations showing $f_{\Delta h}(\mathbf{x})$ (altitude margin) and $f_{\Delta v}(\mathbf{x})$ (allowable speed deviation), representing uncertainties arising from *internal and external factors*. The blue regions indicate vertical speed bounds estimated through forward reachability analysis of the abstracted hybrid automaton model of ArduPilot landing mode controller.

The authors of PGFuzz [8] manually converted such physical behaviors in natural language into the following two MTL formulas (1) and (2):

$$\square\{(\text{Mode}_t = \text{LAND}) \wedge (\text{ALT}_t \geq 10) \rightarrow (\text{Speed\_vertical}_t = \text{WPNAV\_SPEED\_DN})\} \quad (1)$$

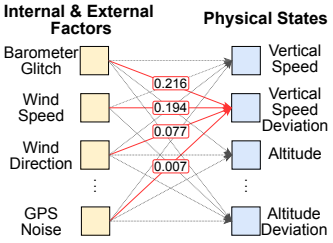
$$\square\{(\text{Mode}_t = \text{LAND}) \wedge (\text{ALT}_t < 10) \rightarrow (\text{Speed\_vertical}_t = \text{LAND\_SPEED})\} \quad (2)$$

While these formulas appear correct and consistent with the documented physical behaviors, they fail to capture *cyber-physical interplay*—specifically, *internal and external factors* that influence an RV's physical states but are not mentioned in the documentation. A drone cannot instantaneously transition from a descent speed of 1.0 m/s to 0.5 m/s at the altitude of 10 meters. Instead, the drone gradually decelerates over a finite time interval, reflecting actuator limitations and aerodynamics. Further, environmental conditions (e.g., wind) affect the drone's descent speed, making it unrealistic to assume a fixed velocity during the landing behavior. In fact, all of the real-world test, abstracted controller model, software-in-the-loop (SITL), and Gazebo simulations consistently demonstrate a 1-meter margin to reach full deceleration ($f_{\Delta h}(\mathbf{x})$ in Fig. 1).

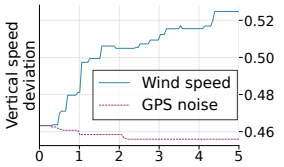
The incorrect specifications above cause frequent false-positive alarms. A false positive in here indicates that an RV's physical states violate MTL formulas, even though the behavior is intentional or not caused by any buggy code line. These two MTL formulas created by PGFuzz lead to 288 false positives during one-hour of running PGFuzz.

To consider *cyber-physical interplay*, RVSPEC (1) identifies the *internal and external factors* that affect an RV's physical states, (2) quantifies how much the identified factors influence those physical states, and (3) create *cyber-physical interplay graphs* (CPGs) based on the quantification results.

The CPG is a directed graph where nodes represent *internal and external factors* (e.g., wind speed), physical states (e.g., vertical speed), and their associated statistical properties (e.g., vertical speed deviation from the configured descent rate). Edges denote quantified influence relationships derived from simulations and/or real-world tests. In Appendix A, we detail



(a) Cyber-physical Interplay Graph.



(b) Vertical speed deviation as wind and GPS noise vary.

Fig. 2: Analysis of how *internal and external factors* affect vertical speed deviation during landing. Edge weights in the CPG reflect importance, e.g., wind speed (0.194) contributing far more than GPS noise (0.007). RVSPEC incorporates these factor-dependent bounds to reduce false positives.

why we do not use model-based analysis (e.g., forward reachability analysis) for the CPG.

For example, in Figure 2, factors (e.g., wind speed, GPS noise) influence the vertical speed during landing behavior. The edge weights reflect the relative strength of these influences, with wind speed assigned a weight of 0.194 and GPS noise a weight of 0.007, indicating that wind plays a greater role in affecting the vertical speed.

We represent *internal and external factors* as a vector $\mathbf{x} = [x_1, x_2, \ldots, x_n]$, where each x_i denotes a specific factor such as wind speed and GPS noise. We define the vertical speed deviation as $\Delta v = |v_{\text{actual}} - v_{\text{expected}}|$, where v_{actual} denotes the actual vertical speed influenced by *internal and external factors* and v_{expected} is the expected speed specified in the documentation. RVSPEC derives two functions with input $\mathbf{x}$: $f_{\Delta v}(\mathbf{x})$ predicts the vertical speed deviation, while $f_{\Delta h}(\mathbf{x})$ adjusts the altitude margin around the 10-meter threshold to decide when the descent phase transitions. RVSPEC applies these functions to generate the parameterized formulas (3) and (4):

$$\square\{(\text{Mode}_t = \text{LAND}) \wedge (\text{ALT}_t \geq 10 + f_{\Delta h_t}(\mathbf{x})) \rightarrow (|\text{Speed\_vertical}_t - \text{WPNAV\_SPEED\_DN}| \leq f_{\Delta v_t}(\mathbf{x}))\} \quad (3)$$

$$\square\{(\text{Mode}_t = \text{LAND}) \wedge (\text{ALT}_t < 10) \rightarrow (|\text{Speed\_vertical}_t - \text{LAND\_SPEED}| \leq f_{\Delta v_t}(\mathbf{x}))\} \quad (4)$$

These formulas incorporate the factor-dependent bounds $f_{\Delta v}(\mathbf{x})$ and the adaptive transition margin $f_{\Delta h}(\mathbf{x})$ derived from the CPG, enabling a more accurate specification of descent behavior during landing. In turn, these two MTL formulas created by RVSPEC reduce the false positives from 288 to 2 during one-hour of running PGFuzz (detailed root causes of the two false positives in Sec. VII-C).

V. RVSPEC

We outline how RVSPEC (1) systematically identifies *internal and external factors* that influence the physical states of RVs, (2) constructs a *cyber-physical interplay graph* (CPG), and (3) leverages a multi-agent LLM framework for specification generation, as shown in Fig. 3.

A. Internal and External Factors Identification

RVSPEC first collects documentation of RV control software and their simulators (e.g., ArduPilot and Gazebo [20]) (1a in Fig. 3), and source code of the corresponding control software (1b). For 1a, RVSPEC conducts documentation

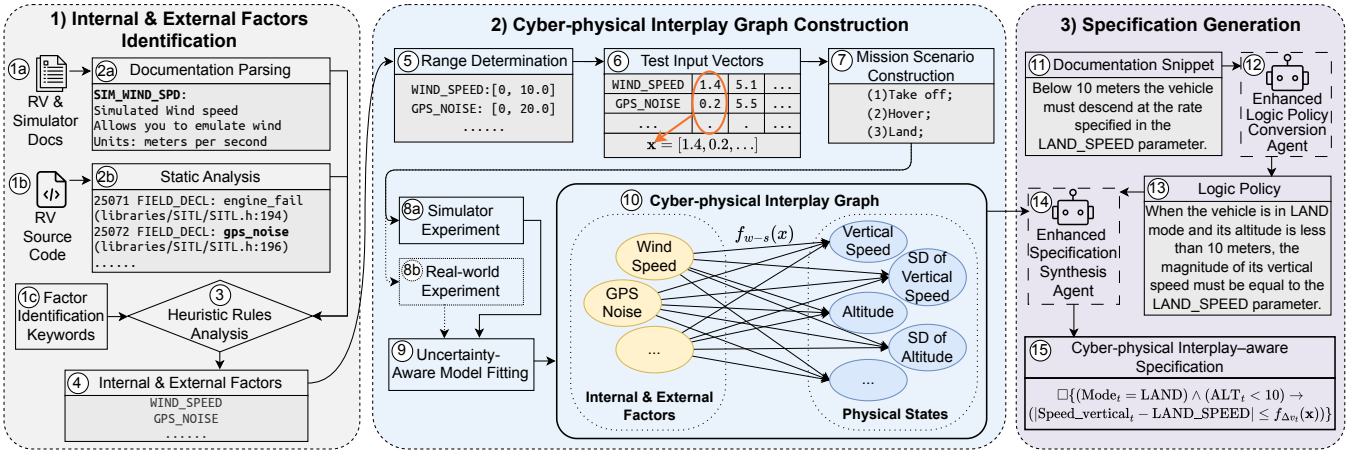


Fig. 3: Overview of RVSPEC's work flow and architecture consisting of three different components: (1) Internal & External Factors Identification, (2) Cyber-physical Interplay Graph Construction, and (3) Specification Generation.

parsing (2a) to extract configuration parameters and variable names. For 1b, it performs static analysis (2b) using LLVM to extract symbolic tables. We also manually construct a domain-specific keyword list (1c) to identify *internal and external factors* that potentially influence RV's physical states, based on the literature review [21]–[24]. RVSPEC then applies heuristic rules (3) to refine the union of candidate factors from 1c, 2a, and 2b, specifically by (*i*) excluding factors related to supplementary features, following a prior work [2], such as logging the flight log to an SD card and (*ii*) merging synonymous terms proposed in PGFuzz [8] (e.g., "acc" and "accel" for acceleration). Next, we conduct manual inspection to finalize the *internal and external factors* (4). One author of this paper spends 13 hours to identify these factors[3] for ArduPilot, PX4, openpilot, and cFS. We note that this task is a one-time effort to run RVSPEC. Yet, users of RVSPEC need to rerun these steps when developers significantly update the control software (e.g., adding new functionality or control algorithm).

B. Cyber-physical Interplay Graph Construction

After identifying the *internal and external factors* (4), RVSPEC determines their value ranges (5). To do so, we manually extract the ranges from sensor datasheets and software configuration references (e.g., gyroscope noise density range is 0–0.0001 rad/s/$\sqrt{\text{Hz}}$ [25]). Yet, control software packages do not have a reference to extract ranges for environmental conditions (e.g., wind, air pressure). Thus, we obtain their typical ranges based on empirical meteorological datasets (e.g. NOAA Climate Dataset [26]).

Given the high dimensionality of the factor space, RVSPEC employs a coverage-guided sampling strategy. In particular, RVSPEC uses latin hypercube sampling (LHS) [27], which divides each factor's range into equal intervals and ensures that each interval is sampled at least once. For example, LHS samples `WIND_SPEED` in $[0.0, 10.0]$ and `GPS_NOISE` in $[0.0, 20.0]$ by selecting one value from each interval with five divisions, resulting in evenly distributed test points. This gives RVSPEC five test points that are evenly spread across the two-dimensional input space. The combinations generated through coverage-guided sampling yield a set of test input vectors (6), each corresponding to a distinct combination of *internal and external factors*.

We construct mission scenarios (7) based on the default missions [28] created by developers and adapt them to different flight modes (e.g., `LOITER`, `CIRCLE`). For instance, a default mission for analyzing landing behavior consists of three phases: (1) takeoff, (2) hover, and (3) land. Then, RVSPEC combines the mission scenarios with the generated test input vectors to perform experiments. During each experiment, RVSPEC records logs, including the RV's physical states and sensor data. Users of RVSPEC can conduct the experiment in either a simulator (8a) or the real world (8b).

After collecting the logs from each experiment, RVSPEC preprocesses the data to derive features for CPG construction. It first extracts relevant physical states (e.g., vertical speed, altitude) from each mission phase (e.g., takeoff, hover, landing), and applies smoothing to reduce noise. Then, RVSPEC computes aggregate statistics (e.g., mean, standard deviation) for each physical state within its corresponding mission phases. RVSPEC uses these processed data to fit uncertainty-aware models (9) that predicts the target physical state based on the given test input vectors (i.e., *internal and external factors*). These models output both the predicted value and interval, estimated using the lower and upper quantiles (e.g., 2.5th and 97.5th). The quantile outputs provide probabilistic bounds on deviations in the physical states. RVSPEC applies 5-fold cross-validation during model training and performs outlier filtering on the evaluation results. We evaluate multiple uncertainty-aware models to select the best model for constructing the final CPG (10) (Detailed in Section VII-D). In the CPG, all *internal and external factors*, physical states, and their statistical features (e.g., mean, standard deviation), are represented as nodes. Directed edges connect factors to statistical features of physical states, with edge weights given by feature importance scores that reflect the relative contribution of each factor. The edge weights are data-driven; thus, they depend on the software version. RVSPEC needs to recompute the weights after each

[3] https://tinyurl.com/5cnjjxre

software update, while the overall pipeline remains reusable.

C. Specification Generation

RVSPEC adopts the concept of a *logic policy* introduced in PGFuzz [8], which serves as an intermediate natural language representation of physical behaviors, specifying temporal relationships between preconditions and postconditions. For instance, given a documentation snippet (⑪) "below 10 meters the vehicle must descend at the rate specified in the `LAND_SPEED` parameter", the *Enhanced Logic Policy Conversion Agent* (⑫) produces the corresponding logic policy (⑬): "When the vehicle is in `LAND` mode and its altitude is less than 10 meters, the magnitude of its vertical speed must equal the `LAND_SPEED` parameter".

To generate such logic policies (⑬), RVSPEC relies on documentation of control software and simulator. If the documentation lacks descriptions of physical behaviors, RVSPEC leverages source code comments as documentation. Then, RVSPEC preprocesses the collected document files by removing non-semantic artifacts (e.g., HTML tags, table formatting characters) and splitting them into semantically coherent sections to respect the input token limitations of LLM agent. Next, the *Enhanced Logic Policy Conversion Agent* (⑫) extracts behavioral descriptions from the preprocessed documentation and translates them into structured logic policies while ignoring non-behavioral descriptions such as motivational context or illustrative examples. These logic policies are then passed to the *Enhanced Specification Synthesis Agent* (⑭), which integrates them with physical constraints derived from the CPG to generate MTL formulas (⑮). These formulas are both semantically aligned with documentation and consistent with physical constraints. To enhance the reliability and accuracy of both agents, we equip them with task-specific few-shot prompts and set the LLM temperature to zero, minimizing randomness, without performing any model fine-tuning.

VI. IMPLEMENTATION

We evaluated RVSPEC on ArduPilot and PX4 with Gazebo, openpilot with Carla [29], and cFS with NOS3 [30].

Simulation Experiments. We wrote 1,703 lines of Python code (LoC) for ArduPilot, 2,126 LoC for PX4, 2,041 LoC for openpilot, and 1,124 LoC for cFS to set up simulation environments and simulate various test scenarios. For ArduPilot *v4.5.7* and PX4 *v1.15.4*, we used Pymavlink *v2.4.42* library to control aerial vehicles in software-in-the-loop (SITL) and Gazebo simulator *v11.15.1*. For openpilot *v0.9.4*, we extended the Python bridge file provided by the developers to support customized configuration and communication with Carla *v0.9.13* simulator. For cFS *equuleus-rc1* and NOS3 *v1.7.3*, we used COSMOS *v5.11.3*, a ground system software, to command and monitor the spacecraft.

CPG Construction. We wrote 3,941 LoC for ArduPilot, 1,947 LoC for PX4, 521 LoC for openpilot, 2,217 for cFS to construct the CPG. To obtain physical states (e.g., vehicle position, velocity), we used flight logs for ArduPilot and PX4, a custom mission log format for openpilot, and telemetry streams and simulator logs for cFS.

Specification Generation. We wrote 1,782 LoC to deploy two enhanced LLM-based agents. We integrated the OpenAI *v1.92.3* and Claude *v0.57.1* APIs to access LLMs and conduct preliminary experiments to design and optimize prompts and few-shot examples for each agent's task.

We conducted all experiments using a desktop machine with Ubuntu 22.04, an Intel i9-10850K@3.60GHz CPU, 32GB RAM, and an NVIDIA RTX 3080 GPU.

VII. EVALUATION

Our evaluation answers the following research questions:
RQ1: How accurately can RVSPEC generate MTL formulas?
RQ2: How effectively do the MTL formulas generated by RVSPEC reduce false positives compared to the baseline?
RQ3: How accurately do CPGs capture the correlation between internal/external factors and an RV's physical states?
RQ4: Can RVSPEC be applied effectively across different RV software packages?

A. Experimental Setup

RVSPEC's Inputs. We collect documentation for control software and simulators, as well as the source code including comments for ArduPilot, PX4, OpenPilot, and cFS. The collected dataset contains 1,436 document files for ArduPilot, 905 for PX4, 20 for openpilot, 43 for cFS. We note that openpilot and cFS do not have thorough documentation; thus, we obtain a smaller number of document files.

LLM Selection. Several LLMs are available for RVSPEC's *Specification Generation* component. To identify the best LLM, we make RVSPEC generate MTL formulas for ArduPilot and PX4, with three different LLMs: gpt-4o (*v2024-11-20*), gpt-4.1 (*v2025-04-14*), and claude-sonnet-4 (*v2025-05-14*). Then, we randomly select 200 MTL formulas per LLM, yielding 600 in total, as evaluation set. The evaluation shows that gpt-4o, gpt-4.1, and claude-sonnet-4 achieve accuracies of 78.5%, 66%, and 82%, respectively. Thus, we decide to use claude-sonnet-4 for RVSPEC and the evaluations.

Datasets. We provide the collected document files and source code, as inputs, to RVSPEC (using claude-sonnet-4). It automatically creates 7,418 MTL formulas for ArduPilot, 3,502 for PX4, 44 for openpilot, and 252 for cFS.

Justification of Using Simulators. To construct CPG, RVSPEC tests various scenarios. We consider three possible approaches: simulation, real-world testing, and model-based analysis (e.g., reachability analysis). We select the simulation approach for the evaluations. It is because the reachability analysis provides the most conservative prediction; thus, it may miss to detect buggy behaviors, as shown in Fig. 1 (Detailed in Appendix A). The real-world tests offer the highest fidelity but is time-consuming, poses safety concerns, and limits control over external factors (e.g., air pressure). In contrast, simulation achieves moderate fidelity and enables full control over *internal and external factors*.

Cyber-Physical Modeling. Various models are available for RVSPEC's *Uncertainty-Aware Model Fitting*. We test seven popular uncertainty-aware models to select the best one (Detailed in Section VII-D). In turn, random forest model

TABLE I: Evaluation Results of Specification Correctness

Software	Syntactic Validity	Semantic Accuracy	Cyber-physical Consistency
ArduPilot	200/200 (100%)	182/200 (91.0%)	164/200 (82.0%)
PX4	200/200 (100%)	181/200 (90.5%)	163/200 (81.5%)
openpilot	44/44 (100%)	39/44 (88.6%)	33/44 (75.0%)
cFS	44/44 (100%)	39/44 (88.6%)	34/44 (77.3%)
Overall	**488/488 (100.0%)**	**441/488 (90.4%)**	**394/488 (80.7%)**

shows the best performance. Thus, we decide to leverage the model for the evaluations.

*B. Specification Correctness (**RQ1**)*

Due to the large number of generated MTL formulas, we randomly sample 200 formulas each from ArduPilot and PX4, 44 formuals each from openpilot and cFS (i.e., a total of 488 formulas). We select 44 because RVSPEC creates the minimum number of formulas (i.e., 44) from openpilot. One author of this paper manually reviews them according to the following criteria: (1) syntactic validity, ensuring the absence of MTL grammar errors; (2) semantic accuracy, verifying that each formula correctly captures the intended physical behaviors as described in the documentation; and (3) cyber-physical consistency, confirming alignment with the expected physical behavior of RVs in the real world. Out of the 488 MTL formulas, 488 satisfy syntactic validity (100%), 441 demonstrate semantic accuracy (90.4%), and 394 show cyber-physical consistency (80.7%), as shown in Table I.

To create a baseline for comparison with RVSPEC, we directly create MTL formulas from documentations used to generate 488 MTL formulas, without leveraging RVSPEC's CPG and multi-agent framework. In turn, this baseline achieves 51.6% accuracy. We also manually analyze the 488 MTL formulas and find that, RVSPEC's CPG can enhance $62/200(31.0\%)$ for ArduPilot, $60/200(30.0\%)$ for PX4, $11/44(25.0\%)$ for openpilot, and $9/44(20.5\%)$ for cFS.

Analysis of Incorrect MTL Formulas. We analyze the 38 incorrect formulas created by RVSPEC. Semantic inaccuracies arise from logical contradictions, overgeneralizations that miss conditions, or incomplete representations of parameter relationships and temporal conditions, e.g., $\square\{(\text{Mode}_t = \text{RETURN}) \rightarrow (\text{Armed}_t = \text{TRUE}) \wedge (\neg\text{Armed}_t = \text{TRUE})\}$, which is a logical contradiction. Cyber-physical inconsistencies stem from the misuse of CPG-derived functions, such as applying a parameterized function to unrelated physical states, e.g., $\square\{|\text{Pitch}_{\text{filtered}_t} - \text{Pitch}_t| \leq f_{\Delta v}(x)\}$, which misuses the vertical speed function for pitch angles.

*C. True Positive and False Positive Rates (**RQ2**)*

We compare the RVSPEC's MTL formulas with those generated by prior work (PGFuzz [8]) as the baseline.

True Positive. We analyze whether the RVSPEC's MTL formulas still detect 156 logic bugs previously identified by PGFuzz. Our results show that RVSPEC's MTL formulas still detect all 156 out of 156 logic bugs, despite employing relaxed thresholds. It is because each of the logic bugs leads to severely deviated physical behaviors (e.g., unstable positions).

False Positive. We apply both the baseline and RVSPEC's MTL formulas and execute one-hour of fuzzing to compare false positive rates. PGFuzz's formulas produce 4,790 false positives across all scenarios, whereas RVSPEC reduces this number to 964 (79.9% reduction), as shown in Fig. 4.

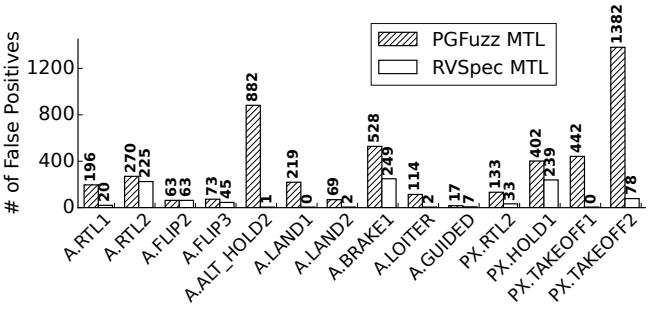


Fig. 4: Comparison of false positives between PGFuzz MTL and RVSPEC's MTL formulas. Each label on the x-axis denotes a distinct MTL formula.

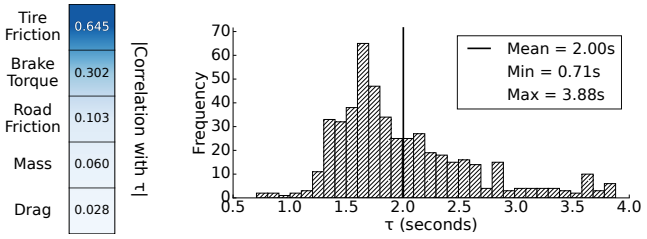
Root Causes of False Positives. We analyze the 964 false positives produced by RVSPEC's MTL formulas and identify three root causes. Out of the 964 cases, 488 (50.6%) result from CPG-derived relaxed constraints that still remain strict under extreme conditions. For example, the CPG specifies a maximum allowable yaw deviation of 0.008, while actual deviations reach 0.012 under severe environmental conditions. In turn, the deviations caused by natural environmental conditions are incorrectly flagged as violations of the MTL formulas. 247 out of the 964 cases (25.6%) arise from overly coarse-grained preconditions in the formulas. For example, `Return-to-Launch (RTL)` flight mode includes both the phase of returning to the home position and landing phase. Relying solely on the precondition: $\text{Mode}_t = \text{RTL}$ fails to differentiate between these distinct phases, leading to normal behavior in one phase being incorrectly identified as anomalous. 229 out of the 964 cases (23.7%) come from the slow update rate of an RV's physical states from the control software and simulators, e.g., altitudes during takeoff are reported as 0.01 at one timestamp and jump to 4.08 at the next. Such a jump is misread as a specification violation.

*D. Cyber-physical Modeling Accuracy (**RQ3**)*

We evaluate how accurately CPGs (based on uncertainty-aware models) capture the correlation between *internal and external factors* and an RV's physical states. To do so, we test performance of seven popular uncertainty-aware models using ArduPilot flight logs collected from both SITL and Gazebo simulators.

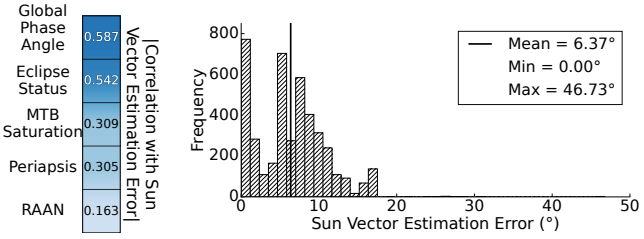
We summarize the performance of each model, reporting the mean and median R^2 across all target physical states, as well as the proportion of targets for which $R^2 \geq 0.7$. Here, the R^2 metric quantifies the proportion of variance in the physical state that is explained by the uncertainty-aware model, with $R^2 = 1$ indicating perfect prediction, $R^2 = 0$ representing no predictive power beyond the mean. We evaluate model performance using 5-fold cross-validation. In turn, random forest achieves the highest overall performance[4], with a median R^2 of 0.5993 and 38.75% of the modeled targets exhibiting $R^2 \geq 0.7$ in SITL simulator datasets.

[4]https://tinyurl.com/yytra5by



(a) (b)

Fig. 5: (a) Top five factors correlated with τ (the time required for a vehicle to completely stop). (b) τ distribution under different internal and external factor combinations.



(a) (b)

Fig. 6: (a) Top five factors correlated with Sun vector estimation error. (b) Sun vector estimation error distribution under different internal and external factor combinations.

The accuracies with the Gazebo simulator are relatively lower than with SITL. The results indicate that Gazebo's physics engine is more complex than SITL's one. Thus, we admit the need for *cyber-physical–aware uncertainty models* to improve the performance.

*E. Generalizability to Other Robotic Vehicles (**RQ4**)*

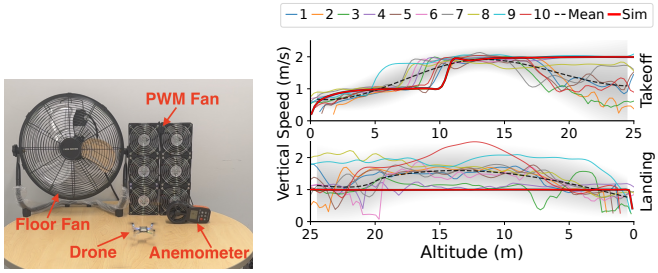
Case Study in Autonomous Vehicles. [7] created 14 temporal logic formulas to formally describe an autonomous vehicle's physical behaviors. RVSPEC's CPG discloses that 12 out of the 14 formulas are physically unrealistic in practice, and refines these 12 formulas to reflect *cyber-physical interplay*. For instance, [7] created a temporal logic formula to express a safe distance between two vehicles: $\square\{\text{TTC}(vv,\ fc) > \tau\}$, where TTC denotes time to collision, τ represents the time required for a vehicle to completely stop, and vv and fc denote two vehicles. Yet, RVSPEC's CPG discovers that the required stopping time τ must vary according to the *internal and external factors* ($\mathbf{x}$), leading to the improved formula: $\square\{\text{TTC}(vv,\ fc) > f_\tau(\mathbf{x})\}$. In particular, the CPG identifies the five most influential factors that affect τ: tire friction, brake torque, road friction, mass, and drag, as shown in Fig. 5a. By testing different combinations of the *internal and external factors*, RVSPEC reveals that mean, minimum, and maximum values of τ are 2.00, 0.71, and 3.88 seconds, respectively, as shown in Fig. 5b. By parameterizing the stopping time (τ) with respect to the relevant *internal and external factors*, the improved MTL formula reduces false positives from 490 to 226 (53.9% reduction) for a one-hour simulation.

Case Study in Spacecrafts. We evaluate RVSPEC on Simulation-To-Flight-1 (STF-1) satellite, running cFS within the NOS3 simulation environment. STF-1 is equipped with one fine sun sensor (FSS) and six coarse sun sensors (CSS) to calculate a Sun vector (i.e., a unit vector pointing from the spacecraft to the Sun) in the `Sunsafe` mode. When the Sun is outside the FSS field of view, the STF-1 relies on the CSSs mounted along six axis-aligned directions to estimate the Sun vector. cFS and NOS3 do not have natural language documentation that describes the expected behaviors of the `Sunsafe` mode. Thus, RVSPEC extracts the following MTL formula from the cFS and NOS3 source code comments: $\square\{(\text{GNS\_MODE}_t = \text{Sunsafe}) \wedge (\text{FSS\_Valid} = \text{False}) \wedge (\text{CSS\_Valid} = \text{True}) \rightarrow (\text{Sun\_Vector}_t = \text{Sun\_Vector}_{\text{CSS}_t})\}$, where Sun_Vector_t is the actual Sun vector, and $\text{Sun\_Vector}_{\text{CSS}_t}$ is the Sun vector calculated from the CSSs. Yet, this formula assumes that the CSSs always return the true Sun vector with no measurement error (i.e., $|\text{Sun\_Vector}_t - \text{Sun\_Vector}_{\text{CSS}_t}| = 0$), even though the CSS readings include sensor noises in practice. For instance, the Earth albedo effect is the reflection of sunlight from the Earth's surface and atmosphere, which affects the accuracy of CSS measurements. In fact, RVSPEC's CPG discovers the five most influential factors affecting the accuracy of CSS readings (i.e., Sun vector estimation errors): (1) global phase angle (the geometric Sun–Earth–spacecraft angle), (2) eclipse status, (3) magnetorquer bar (MTB) saturation, (4) periapsis, and (5) right ascension of the ascending node (RAAN), as shown in Fig. 6a. All five factors, except MTB saturation, determine the impact of the Earth albedo effect. By testing different combinations of the *internal and external factors*, RVSPEC finds that a mean sun vector estimation error is 6.37°. We note that we use direction angles to represent the vector error here; thus, the unit is degrees, as shown in Fig. 6b. By parameterizing the Sun vector estimation errors with respected to the identified *internal and external factors*, the refined formula becomes: $\square\{(\text{GNS\_MODE}_t = \text{SUNSAFE}) \wedge (\text{FSS\_Valid} = \text{False}) \wedge (\text{CSS\_Valid} = \text{True}) \rightarrow (|\text{Sun\_Vector}_t - \text{Sun\_Vector}_{\text{CSS}_t}| \leq f_{\Delta sun\_vector_t}(\mathbf{x}))\}$. The improved MTL formula reduces false positives from 2,372 to 1,353 (43.0% reduction) in a one-hour simulation where the CSSs return sensor readings under the Earth albedo effect.

F. Measuring Sim-to-Real Gap

We conduct real-world experiments, as shown in Fig. 7a, to validate our approach and quantify the sim-to-real gap. The experimental setup consists of a PWM-controlled fan and a 14-inch high-velocity floor fan as wind sources, generating wind speeds ranging from 0.5-4.2 m/s and 4-6.5 m/s, respectively. We use a Bitcraze Crazyflie 2.1+ quadcopter as the test platform and use an anemometer to measure real-time wind speed during the experiments.

We analyze 10 real-world flight logs and one flight log generated by PX4 with Gazebo simulator, all conducted under the same takeoff and landing target speed (1 m/s), as shown in Fig. 7b. We compute correlations across the takeoff and landing phases: the average correlation among real-world logs is 0.701 for takeoff and 0.390 for landing, while the average correlation between real-world and simulation logs drops to 0.646 and 0.147, respectively. A t-test confirms that the internal real-world correlations are significantly higher



(a) (b)

Fig. 7: (a) Real-world experiments setup. (b) Vertical speed in takeoff and landing for 10 real-world tests and simulation.

than the real-vs-simulation correlations during landing ($p = 0.0034$), but not significant during takeoff ($p = 0.586$).

VIII. Conclusions and Future Work

We reveal that formal specifications require an understanding of how *internal and external factors* affect an RV's physical states. To do this, we introduce RVSpec, a framework that first constructs a *cyber-physical interplay graph* (CPG) and then leverages both documentation and the CPG to automatically generate *cyber-physical interplay–aware specifications*. We evaluate RVSpec on four popular RV control software for aerial vehicles, autonomous vehicles, and spacecrafts. Compared to a 51.6% baseline, RVSpec achieves 80.7% correctness and reduces false positives by 79.9%, while maintaining bug-finding capability.

We admit a few limitations: (1) uncertainty model-based CPGs offer statistical rather than formal guarantees, and (2) LLMs still lack a deep understanding of control software; thus, poor documentation can lead RVSpec to generate incorrect specifications. Our future work will focus on (*i*) developing *cyber-physical–aware uncertainty models* and (*ii*) automated state machine generation to understand control software.

References

[1] Japan's moon lander failure, https://tinyurl.com/ycker32r, 2023.

[2] H. Kim, M. O. Ozmen, Z. B. Celik, A. Bianchi, and D. Xu, "PatchVerif: Discovering Faulty Patches in Robotic Vehicles," in *Proceedings of the USENIX Security Symposium*, 2023.

[3] ——, "PGPATCH: Policy-guided logic bug patching for robotic vehicles," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2022.

[4] "ArduPilot," https://github.com/ArduPilot/ardupilot_wiki, 2025.

[5] "PX4," https://github.com/PX4/PX4-user_guide, 2025.

[6] E. M. Clarke, "Model Checking," in *Proceedings of the International conference on foundations of software technology and theoretical computer science*, 1997.

[7] R. Song, M. O. Ozmen, H. Kim, R. Muller, Z. B. Celik, and A. Bianchi, "Discovering adversarial driving maneuvers against autonomous vehicles," in *Proceedings of the USENIX Security Symposium*, 2023.

[8] H. Kim, M. O. Ozmen, A. Bianchi, Z. B. Celik, and D. Xu, "PGFUZZ: Policy-Guided Fuzzing for Robotic Vehicles," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2021.

[9] L. Lamport, "The temporal logic of actions," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 1994.

[10] "F Prime," https://fprime.jpl.nasa.gov/, 2025.

[11] R. L. Bocchino, J. W. Levison, and M. D. Starch, "FPP: A Modeling Language for F Prime," in *Proceedings of the IEEE Aerospace Conference (AERO)*, 2022.

[12] Y. Chen, R. Gandhi, Y. Zhang, and C. Fan, "NL2TL: Transforming Natural Languages to Temporal Logics using Large Language Models," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

[13] "Openpilot," https://comma.ai/openpilot, 2025.

[14] "The Core Flight System (cFS)," https://github.com/nasa/cFS, 2025.

[15] S. Zhang, J. Zhai, L. Bu, M. Chen, L. Wang, and X. Li, "Automated Generation of LTL Specifications For Smart Home IoT Using Natural Language," in *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020.

[16] J. He, E. Bartocci, D. Ničković, H. Isakovic, and R. Grosu, "DeepSTL - From English Requirements to Signal Temporal Logic," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2022.

[17] K. Manas, S. Zwicklbauer, and A. Paschke, "TR2MTL: LLM based framework for Metric Temporal Logic Formalization of Traffic Rules," in *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*, 2024.

[18] H. Li, Z. Dong, S. Wang, H. Zhang, L. Shen, X. Peng, and D. She, "Extracting Formal Specifications from Documents Using LLMs for Automated Testing," in *Proceedings of the International Conference on Program Comprehension (ICPC)*, 2025.

[19] "ArduPilot Land Mode," https://tinyurl.com/5ev3nwks, 2025.

[20] "Gazebo," https://gazebosim.org/docs/latest/getstarted/, 2025.

[21] M. Gao, C. H. Hugenholtz, T. A. Fox, M. Kucharczyk, T. E. Barchyn, and P. R. Nesbit, "Weather constraints on global drone flyability," *Scientific Reports*, 2021.

[22] B. H. Wang, D. B. Wang, Z. A. Ali, B. Ting Ting, and H. Wang, "An overview of various kinds of wind effects on unmanned aerial vehicle," *Measurement and Control*, 2019.

[23] G. Zhao, L. Liu, S. Li, and S. Tighe, "Assessing pavement friction need for safe integration of autonomous vehicles into current road system," *Journal of Infrastructure Systems*, 2021.

[24] J. Vargas, S. Alsweiss, O. Toker, R. Razdan, and J. Santos, "An overview of autonomous vehicles sensors and their vulnerability to weather conditions," *Sensors*, 2021.

[25] "ICM-20689 Datasheet," https://tinyurl.com/y5ju3t47, 2025.

[26] "NOAA Climate Data," https://tinyurl.com/mpakpkh, 2025.

[27] M. D. McKay, R. J. Beckman, and W. J. Conover, "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code," *Technometrics*, 2000.

[28] "Copter Mission," https://tinyurl.com/2hshuec9, 2025.

[29] "Carla," https://carla.org/, 2025.

[30] "NASA Operational Simulator for Small Satellites (NOS3)," https://github.com/nasa/nos3, 2025.

[31] H. Park, V. Vijay, and I. Hwang, "Data-driven reachability analysis for nonlinear systems," *IEEE Control Systems Letters*, 2024.

[32] L.-Y. Lin, J. Goppert, and I. Hwang, "Log-linear dynamic inversion control with provable safety guarantees in lie groups," *IEEE Transactions on Automatic Control*, 2024.

[33] J. V. Deshmukh and S. Sankaranarayanan, "Formal Techniques for Verification and Testing of Cyber-Physical Systems," *Design Automation of Cyber-Physical Systems*, 2019.

Appendix

A. Model-Based CPG Generation

RVSpec could also use model-based methods such as forward reachability analysis [31] and control invariance analysis [32] to construct CPGs. They provide theoretically rigorous ways to define uncertainty-aware constraints. Yet, we decide to select simulation and real-world tests due to the following two challenges and evaluation results. First, they lack scalability; as system complexity and the number of operation modes (e.g., flight modes) increase, they become computationally infeasible and often yield overly conservative results for control software [33]. Second, they require complete white-box models; yet, capturing all low-level details is infeasible, causing the resulting sets to deviate from actual behavior and leading to both false positives and missed buggy behaviors. Indeed, as shown in Fig. 1, the reachable set bounds from forward reachability analysis of the drone's dynamic model are sound but overly conservative, encompassing real-world measurements with margins too large for direct use in constructing CPGs.